VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Rychlá implentace analytického programování pro rozsáhlé datasety

# Fast implementation of Analytic Programming used for large amount of datasets

2014 Bc. Peter Drábik

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

# Diploma Thesis Assignment

Student: **Bc. Peter Drábik**

Study Programme: N2647 Information and Communication Technology

Study Branch: 2612T025 Computer Science and Technology

Title: Rychlá imlementace analytického programování pro rozsáhlé datasety
Fast implementation of Analytic Programming used for large amount of datasets

Description:

The main goal of this master thesis is the design, development and testing of a program implementing Analytic Programming a novel method that allows solving problems from the symbolic regression domain.

The main tasks to do in the thesis are:

1. State of the art in the Analytical programming area and nowadays parallel architecture based on CUDA.
2. Design of an architecture of fast and scalable building of fitting function for given data, using analytical programming.
3. Implementation of previously defined method and case study for given datasets.
4. Prepare the performance test on a selected set of data mining benchmarks including astronomical problems. Comparison of various evolutionary algorithms used with analytic programming (e.g. SOMA, Differential Evolution). Follow international standards, such are DAME and IVO (International Virtual Observatory), to be compatible.
5. Preparation of programmer and user's documentation and datasets for benchmarking, or comparison with your results.

References:

[1] Zelinka, I., Oplatkova, Z., Nolle, L.: Analytic programming —Symbolic regression by means of arbitrary evolutionary algorithms. Int. J. of Simulation, Systems, Science and Technology 6(9), 44–56 (2005)
[2] Zelinka, I., Oplatkova, Z.: Analytic programming — Comparative study. In: Proc. the Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, Singapore, paper No. PS04-2-04 (2003)
[3] Zelinka, I.: Analytic programming by Means of new evolutionary algorithms. In: Proc. 1st International Conference on New Trends in Physics 2001, Brno, Czech Republic, pp. 210–214 (2001)
[4] Farber, R.: CUDA Application Design and Development, Morgan Kaufmann(2011)

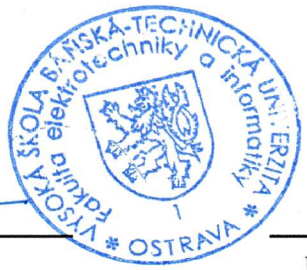
Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. RNDr. Petr Šaloun, Ph.D.**

Consultant: RNDr. Petr Škoda, CSc.

Date of issue: 16.11.2012

Date of submission: 07.05.2014




doc. Dr. Ing. Eduard Sojka
*Head of Department*

prof. RNDr. Václav Snášel, CSc.
*Dean of Faculty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2014 .....

I wish to express my appreciations and gratitude to doc. RNDr. Petr Šaloun, Ph.D. and prof. Ing. Ivan Zelinka, Ph.D. for their help, ideas and time.

## Abstrakt

Tato práce se zaobírá metodou, kterou lze dobře využít pro analýzu spekter – analytickým programováním a jeho rychlou realizací. Mým cílem je vytvořit matematické vzorce emisních čar ze spekter, která jsou charakteristická pro Be hvězdy. Jedním z možných řešení tohoto úkolu je symbolická regrese, která v naší aplikaci představuje proces, kdy naměřená data jsou namodelované na nejlépe reprezentující matematický vzorec. V současné době existují počítačové metody, které nám umožňují provádět tyto výpočty více, či méně efektivně. Novou metodou v symbolické regresi, ve srovnání s genetickým programováním a gramatickou evolucí, je analytické programování. Cílem této práce je ověřit efektivitu paralelního přístupu tohoto algoritmu, pomocí CUDA architektury, který lze spustit na serveru. Dále se budu zabývat realizací náhodných rozhodovacích lesů, které mohou být požité ke klasifikování obrovského množství různých spekter s pomocí matematických funkcí získaných pomocí analytického programování, jak je uvedeno v malém příkladu.

**Klíčová slova:** symbolická regrese, analytické programování, evoluční algoritmy, paralelní programování, deterministický chaos, CUDA, SOMA, DE

## Abstract

The aim of this master thesis is to discuss a method useful for spectra analysis – analytical programming and its fast implementation. My goal is to create mathematical formulas of emission lines from spectra, which are characteristic for Be stars. One issue in performing this task is symbolic regression, which represents the process in our application, when measured data fit the best represented mathematical formula. In past this was only a human domain; nowadays, there are computer methods, which allow us to do it more or less effectively. A novel method in symbolic regression, compared to genetic programming and grammatical evolution, is analytic programming. The aim of this work is to verify the efficiency of the parallel approach of this algorithm, using CUDA architecture, which can be run on a server. Next I will discuss implementation of random decision forest to classify huge amounts of various spectra with the help of mathematical functions obtained via analytical programming, as shown in small example.

**Keywords:** symbolic regression, analytic programming, evolutionary algorithms, parallel programming, deterministic chaos, CUDA, SOMA, DE

## Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| AP | – | Analytical Programming |
| API | – | Application Programming Interface |
| ARFF | – | Attribute-Relation File Format |
| BNF | – | Backus–Naur Form |
| BOAT | – | Bootstrapped Optimistic Algorithm for Tree construction |
| CART | – | Classification And Regression Trees |
| CPU | – | Central Processing Unit |
| CUDA | – | Compute Unified Device Architecture |
| DAL | – | Data Access Layer |
| DAME | – | DAta Mining and Exploration |
| DE | – | Differential Evolution |
| DSH | – | Discrete Set Handling |
| DT | – | Decision Tree |
| EA | – | Evolutionary Algorithm |
| FITS | – | Flexible Image Transport System |
| GE | – | Grammatical Evolution |
| GP | – | Genetic Programming |
| GPU | – | Graphics Processing Unit |
| HTTP | – | HyperText Transfer Protocol |
| ID3 | – | Iterative Dichotomiser 3 |
| IVOA | – | International Virtual Observatory Alliance |
| PCIe | – | Peripheral Component Interconnect Express |
| PRNG | – | Pseudo Random Number Generator |
| RDF | – | Random Decision Forests |
| REST | – | Representational State Transfer |
| SIMT | – | Single Instruction Multiple Threads |
| SOMA | – | Self Organizing Migrating Algorithm |
| VO | – | Virtual Observatory |

# Contents

# List of Tables

# List of Figures

# Seznam výpisů zdrojového kódu

# 1 Introduction

Nowadays, many scientific disciplines produce enormous amount of data. Every new technology increases parameter space or allows better sampling. Handling, processing and modelling of this huge amount of data present a major challenge for IT and forms an important part of computational problems. New science disciplines such as astroinformatics has emerged from this technology driven process. These disciplines are using the newly created tools - virtual observatory, machine learning, grid computing, data mining.

This work deals with the astroinformatics problem of finding the best and the fastest method useful for spectra analysis, to create mathematical formulas of extracted emission lines from spectra, characteristic for Be stars.

Be stars are hot B-type stars (effective temperature 10,000 to 30,000K) with luminosity class III to V (i.e. not supergiant stars) whose spectrum has shown at least once an emission line – usually hydrogen in the Balmer line. Sometimes, other emission lines are visible, for example neutral helium. Even when the spectrum goes back to *normal*, the star remains in the Be star class [1]. Some of them are among the brightest stars in the sky [2].

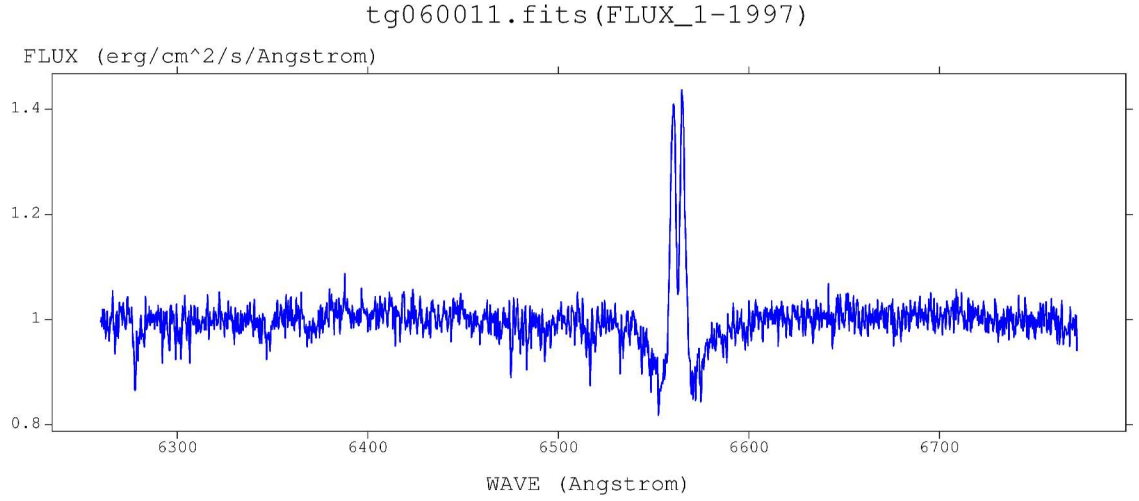



Figure 1: shows an example of spectra with a characteristic Be star emission.

The first chapter describes the data formats used to store spectra informations. The aim is to create an application, which reach requirements to be a model plugin in DAMEWARE, which is described in chapter 2, 3. I decide to use for classification of the result of analytic programming random decision forests, described in chapter 4. In chapter 5 I describe evolutionary algorithms which I am using in analytical programming. Chapter 6 describes symbolic regression with its methods which I am focusing on to solve problem of obtaining mathematical formulas of spectra. I am focusing to novel method - analytical programming which I describe more complex in this work. Since the main objective is to implement as quick algorithm as possible I will try to benefit from parallel architecture

based on CUDA, which I describe in chapter 7. Then I will try to improve performance and convergence with deterministic chaos and discuss measured results in chapter 8.

For handling heterogeneous data distributed data is necessary to define set of standards and protocols. Authority ensuring this job is called International Virtual Observatory Alliance (IVOA). Standards and specifications produced by IVOA can be found at `http://www.ivoa.net/`. Currently it includes 21 member institutions.


Figure 2: IVOA logo [3]

# 2 Virtual Observatory

Virtual observatory is a collection of interoperating data archives and tools through internet and it creates scientific environment for researches, which can handle different astronomical researches.

## 2.1 Architecture Overview

The objective of the Virtual Observatory is shown at Figure 3 to improve and unify access to astronomical data and services primarily for professional astronomers, but also for the general public. The top bar of the figure represents this objective: discovery of data and services, reframing and analysing that data through computation, publishing and dissemination of results, and increasing scientific output through collaboration and federation. The IVOA does not specify or recommend any specific portal or library by which users can access VO data, but some examples of these portals and tools are shown in the grey box [3] . To obtain data from VO we must follow IVOA standards for data access layer (DAL). DAL services are provided as HTTP REST[1] webservices. For spectra analysis are interesting following services:

- *Simple Spectral Access Protocol* - defines an uniform interface to remotely discover and access one-dimensional spectra and aggregations of 1-D spectra. Discovered datasets are returned in VOTable format
- *Simple Line Access Protocol* - defines interface to spectral lines search from Spectral Line Data Collections. Spectral lines are returned in VOTable format,Title Suppressed Due to Excessive Length Simple Image Access Protocol allows to retrieve on-the-fly created images of the sky given the position and the size of the desired output image. After client has described image, service returns list of possible images in VOTable format.Client then chooses image to retrieve. Retrieved image is stored in FITS format
- *Simple Cone Access Protocol* - protocol to retrieve records from a catalogue of astronomical sources. The query describes sky position and an angular distance, defining a cone on the sky. The response returns a list of astronomical sources from the catalog whose positions lie within the cone, formatted as a VOTable.

To obtain spectra and metadata of spectra to process is needed to use these protocols.

## 2.2 Data Formats

Astronomical data such as spectra needs to be stored in standardized formats, for ease later processing. Virtual observatory takes advantage from the few very good standards of data formats, which are implemented in various languages and virtual observatory implements its own, based on these standards.

[1] `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`

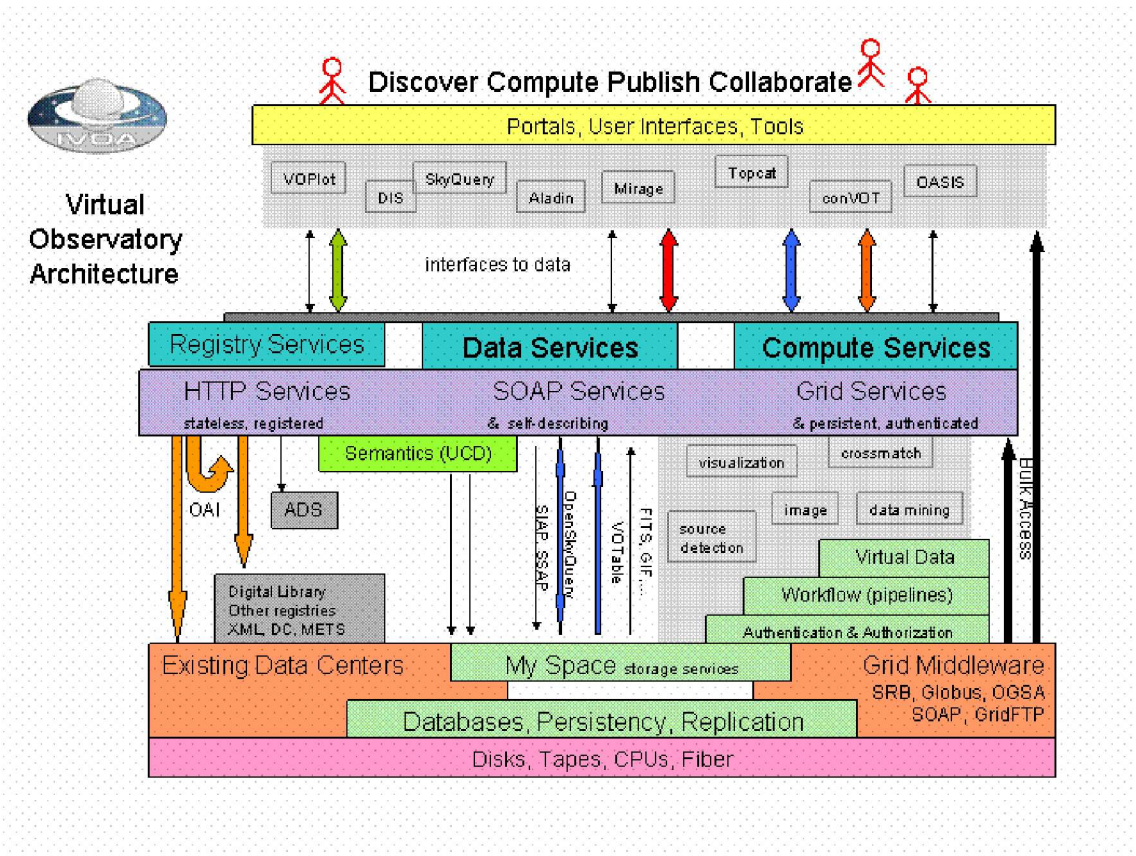



Figure 3: Virtual observatory architecture

### 2.2.1 Flexible Image Transport System

Non-image data, such as spectra, are also mainly distributed in astronomy in Flexible Image Transport System (FITS) format, standardized in 1981 [4]. FITS (Flexible Image Transport System) was originally created for data exchange between WSRT 2 and the VLA 3 [5]. It is now used as a file format to store, manipulate and transmit not only image, but almost all scientific data. This format can have the `*.fts`, `*.fits`, `*.fit` extensions and a major feature it has is that images metadata are store in a human readable ASCII header. In this work I processed fits format data files with fv FITS Editor, available from the standard Ubuntu repository.

### 2.2.2 VOTable

The VOTable format is an XML standard for representing a set of tables, aiming at exchanging properly described data between agents acting in the framework of the Virtual Observatory [3]. This format has features for big data and grid computing. Big data can be referenced with URL, smaller as simply XML.

### 2.2.3 Other Formats

The IVOA architecture will also define some other data-structure formats. One example is the Array structure, representing a subscript-indexed set of voxels in n dimensions, where each voxel value has the same primitive type [3].

# 3 DAta Mining and Exploration

DAta Mining and Exploration (DAME)[2] is an Italian project to provide Astrophysics community with easy to use a data mining suit. This suit is called DAMEWARE. Its focus is on processing of massive data sets with machine learning methods. It is based on S.Co.PE.[3] , a general-purpose supercomputing infrastructure of the University of Naples Federico II. DAMEWARE model library is extendable via plugins. The only requirement is to have executable of model and to supply parameters for model or configuration files without need to know underlying DAMEWARE architecture. The aim is to create an application, which reach requirements to be a model plugin in DAMEWARE.

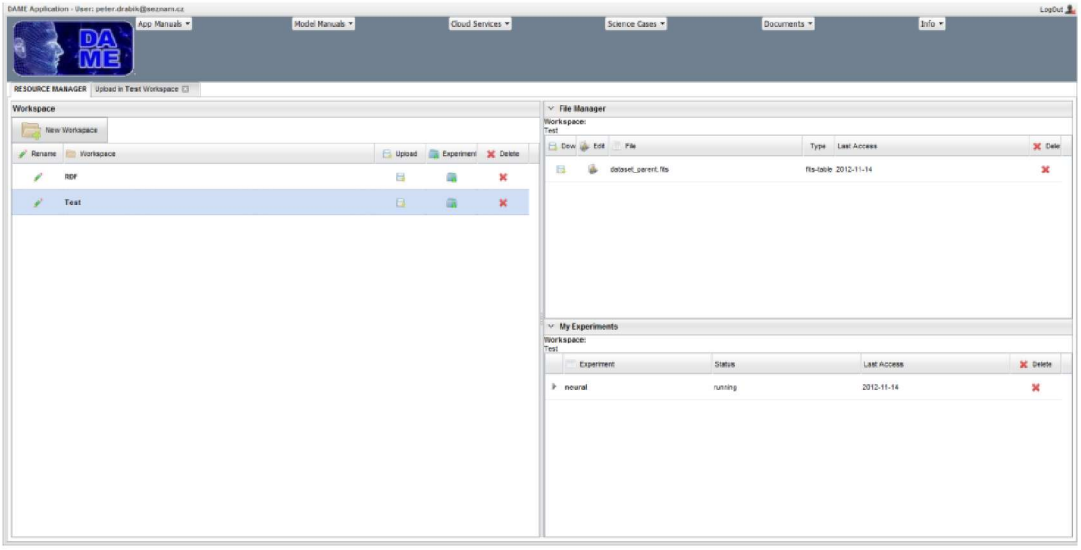



Figure 4: DAMEWARE workspace

[2] http://dame.dsf.unina.it
[3] http://www.scope.unina.it

# 4 Data Mining with Random Decision Forests

Labelled data are used to predict the value of that attribute for instances that have not been seen yet. Data mining using labelled data is known as supervised learning. If the designated attribute is categorical, the task is called classification. If the designated attribute is numerical, the task is called regression [6].

Classification is the process of finding a model (or function) that describes and distinguishes data classes or concepts. The model are derived based on the analysis of a set of training data (i.e., data objects for which the class labels are known). The model is used to predict the class label of objects for which the the class label is unknown. The derived model may be represented in various forms, such as classification rules (i.e., IF-THEN rules), decision trees, mathematical formulae, or neural networks. A decision tree (DT) is a flowchart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. DT can easily be converted to classification rules [7].

## 4.1 The Construction of the Predictive Model

We use an algorithm called *tree inducer* for learning DT. These inducers are constructing DT automatically from given dataset. The aim is usually to find the optimal DT by minimizing the generalization error. We can use the following algorithms, as an example: ID3, C4.5, CART, CHAID, RainForest, BOAT and more.

- ID3, C4.5, and CART adopt a *"greedy"* approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built [7].

- Bootstrapped Optimistic Algorithm for Tree construction (BOAT) - is a decision tree algorithm that uses a statistical technique known as "bootstrapping" to create several smaller samples (or subsets) of the given training data, each of which fits in memory. Each subset is used to construct a tree, resulting in several trees. BOAT usually requires only two scans of dataset. This is quite an improvement, even in comparison to traditional decision tree algorithms, which require one scan per tree level. BOAT was found to be two to three times faster than RainForest, while constructing exactly the same tree [7].

## 4.2 Experiments with Decision Trees

Experiments in [8] over the datasets with the number of attributes in the approximate range of 750 to $700K$ showed that random decision forests are better behaved at very high dimensions, and it is easy to parallelize scales efficiently to high dimensions.The results were confirmed in experiments [9], where boosted DT were more effective when the dimensionality was low. This study recommends the choice of boosted DT up to 4000

dimensions. Best performance above this number was achieved with random decision forests. This method has been successfully used to solve astronomical problems [10, 11].

## 4.3 Random Decision Forests

Random Decision Forests (RDF) were introduced by Ho (1995) [12]. RDF method is based on a collection of DTs, built with some elements of randomisation. This method needs a training dataset. There is present any tree pruning in the RDF. Instead, the RDF create a set of highly uncorrelated trees and combine their results in order to create a generalized predictor. RF are a combination of tree predictors, where each tree depends on the values of a random vector sampled independently and they have the same distribution for all trees in the forest. Generalization error for forests converges so much to the limit, as the number of trees in the forest increases. Error of tree classifiers in forest depends on the strength of the individual trees and the correlation between them. Using a random selection of features to split each node lower the error rate, which stands up well in comparison with AdaBoost[4], but it is more resistant to noise [13].

## 4.4 Software with implemented Random Decision Forests

- Random Forests®[5] - Random Forests(tm) is a trademark of Leo Breiman and Adele Cutler and is licensed exclusively to Salford Systems.
- Waffles[6] - is trying to be the world's most extensive collection of command line tools for machine learning and data mining. It's implemented in C++.
- R[7] - the environment for statistical computing and graphics. Contains package randomForest. R provides an interface to the original programs in Fortran.
- Rf-ace[8] - It is the effective implementation of robust machine learning algorithms. Rf-ace is written in C++. It includes effective implementation of the RDF.
- Weka[9] - It is an open-source collection of machine learning algorithms, developed at the University of Waikato in New Zealand. The program is written in Java.

[4] AdaBoost has no random elements and the set of trees is growing with balancing of a training dataset, where the current weight depended on the previous formation of the file.

[5] Proprietary software. Available at: `http://www.salford-systems.com/en/`

[6] Freeware. Available at: `http://sourceforge.net/projects/waffles/files/`

[7] Freeware. Available at: `http://cran.r-project.org/`

[8] Freeware. Available at: `http://code.google.com/p/rf-ace/downloads/list`

[9] Freeware. Available at: `http://www.cs.waikato.ac.nz/ml/weka/`

# 5 Evolutionary Algorithm

Many technical problems can be formulated as optimization problem. Typical examples are finding of best trajectory for robots, finding of best shape of the antenna and many others. We convert these problems to mathematical problems with the appropriate functional transcription and we are trying to find best arguments of fitness function. We want usually to reach global extreme, most frequently to find global minimum.

It has been developed special class of algorithms called evolutionary algorithms to deal with these types of problems. Their domain may differs (integer, real, complex numbers,...). Within the optimization can be applied various penalties and restrictions not only to the arguments, but also to value of fitness function.

For evolutionary algorithms is typical that they work with population, which consists from individuals. These individuals impact each other and they may change over the iterations. The iterations are usually called generations or migrations. Typical example of evolutionary algorithms, I use in this work, are Self Organizing Migration Algorithm and Differential Evolution. From other algorithms I can mention Simulated Annealing.

## 5.1 Differential Evolution

Differential evolution (DE) was proposed in 1995 by Ken Price and Rainer Storm. This evolutionary algorithm is comparable to genetic algorithms and it has several similar features as generations, creation of population, etc.

In principle, it is based on the so-called genetic annealing, which was later on changed by switching representation from the binary to the decimal system and by changing logic operations to vector operations. By means of these modifications, an algorithm suitable for numeric optimization was created. The DE was later enhanced when the so-called differential mutation was found. [14]

### 5.1.1 Parameters of Differential Evolution

- $NP$ - The Population Numbers - The size of population decides the total number of the solution vectors in the same iteration. Large population required longer computing time, but too small population in a search of a large solution. Size of population should not be less than 4, because it's a minimal value, when DE is still working.

- $F$ - The Mutation Factor - This factor decides how many perturbation ratios the solution can acquire. If the value is greater, the magnitude of jump will increase. In other words, population can break away the regional optimum effectively. Small mutation factor can converge rapidly, but has a high probability of dropping in a regional optimum.

- $CR$ - The Crossover Rate - A key parameter affecting the operation of DE is the crossover rate. While very low values are recommended for and used with separable problems, on non-separable problems, which include most real-world problems,

$CR = 0.9$ has become the standard, working well across a large range of problem domains.

- $Generations$ - Indicates the number of evolutionary cycles( generations), in which the whole population is generally cultivated.
- $D$ - The dimension - parameter determines the number of parameters in the object function.

After many experimental analysis, Gämperle et al. [15] recommended that a good choice for $NP$ is between $3D$ and $8D$, with $F = 0.6$ and $CR$ lies in $< 0.3, 0.9 >$. Contrarily, Rönkkönen et al. [16] concluded that $F = 0.9$ is a good compromise between convergence speed and convergence probability. Additionally, $CR$ depends on the nature of the problem, so $CR$ with a value between $0.9$ and $1$ is suitable for non-separable and multimodal objective functions, while a value of $CR$ between $0$ and $0.2$ when the objective function is separable [17].

| **Control variables** | **Interval** | **Best?** | **Comments** |
|---|---|---|---|
| $NP$: Population | $< 10D; 100D >$ | $10D$ | $100D$ for very multimodal function |
| $F$: Scaling factor | $< 0; 2 >$ | $0.3 - 0.9$ | |
| $CR$: Crossover probability | $< 0; 1 >$ | $0.8 - 1.0$ | $CR = 0$ function is separable, $CR = 1$ function is epistatic |
| $Generations$ | User defined | | |
| $D$: Dimension | Problem-specific | | Number of objective function arguments |

Table 1: Operating parameters for DE [18]

### 5.1.2 The Sequence of Differential Evolution's Steps

Main principle of DE, is shown in algorithm below:

```
 Initialization
Evaluation
repeat
   Mutation
   Crossover
   Evaluation
   Selection
until Stopping criterion ==true
```

Listing 1: Main parts of DE algorithm

1. *Initialization* - In initialization phase is necessary to specify the parameters characterizing boundary of searching space. The convenient way is to collect this data in two D-dimensional vectors where subscripts indicate the lower and upper bounds

respectively. Once these parameters have been predefined, the initial population can be constructed using random number generator under following formula [19]:

$$(x_{i,j,0}) = randj(0,1) * (b_{j,U} - b_{j,L}) + b_{j,L},$$
$$i = 0, 1, \ldots, NP - 1,$$
$$j = 0, 1, \ldots, D - 1.$$

2. *Mutation* - Every specific type of EA has its own mechanism of mutation. The procedure of mutation is applied for every population member in current population $P_{x,g}$ to produce intermediate population $P_{v,g}$. DE algorithm got his name from differential mutation, which for the current generation has the following:

$$v_{i,g} = x_{r0,g} + F * (x_{r1,g} - x_{r2,g}),$$
$$i = 0, 1, \ldots, NP - 1.$$

where $F$ is the scale factor controlling evolvement of population. It is a positive real number and although there is no upper limit of this constant, effective value is rarely greater than one. One can distinguish three types of vector contained in this formula, they are target vector with index $i$, base vector with index $r_0$ and difference vectors with index $r_1$ and $r_2$. $r_0$, $r_1$ and $r_2$ are three different randomly chosen numbers corresponding to indexes in current population which differ from index of target vector [19].

3. *Crossover* - Crossover is essential part of every EA methods, which is responsible for recombination of current with mutant population to produce trial population. Similarly to mutation, crossover mechanism is applied for every element in intermediate population ($i = 0, 1, \ldots, NP-1, j = 0, 1, \ldots, D-1$) in current generation $g$. Hence, the output trial population has the same size as the current population. Due to this circumstance the selection step of DE algorithm can be conducted element wise. Classical DE provides uniform crossover procedure according following formula:

$$u_{i,g} = u_{i,j,g} = \begin{cases} v_{i,j,g}, & \text{if } (randj(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{i,j,g}, & \text{otherwise.} \end{cases}$$

Thus, Parameter $Cr$ here is the crossover probability which is user- defined parameter controlling fraction of parameters inherited from the mutant.In order to determine which source contributes a given parameter, a uniformly random-generated number is compared with the crossover probability. Also, additional condition ($j = j_{rand}$) ensures that trial vector differs from current vector [19].

*Selection* - Procedure of surviving the fittest is realized in DE according to the value of objective functions for given trial and current vectors. Thus, every trial vector is compared with current vector element wise and in the case when objective function value for trial vector is lower or equal to such value of current vector, it replaces the

current vector in next generation. This procedure can be explained by the following formula, where $i = 0, 1, \ldots, NP - 1, j = 0, 1, \ldots, D - 1$:

$$x_{i,g+1} = \begin{cases} u_{i,g}, & \text{if } f(u_{i,g}) \leq f(x_{i,g}) \\ x_{i,g}, & \text{otherwise.} \end{cases}$$

Once the next generation is fully constructed, whole process starting from mutation to selection is repeated with selected population until the optimum is located or specified termination condition is satisfied [19].

### 5.1.3 Versions of Differential Evolution

Differential evolution offers a whole score of versions that mainly differ in how they calculate the noise vector and compose the testing vector after that.
Samples of DE versions [14]:
DE/best/1/exp

$$v_j = x^G_{best,j} + F * (x^G_{r2,j} - x^G_{r3,j})$$

DE/rand/1/exp

$$v_j = x^G_{r1,j} + F * (x^G_{r2,j} - x^G_{r3,j})$$

## 5.2 Self Organizing Migrating Algorithm

The Self Organizing Migrating Algorithm (SOMA)is a relatively new (1999) stochastic evolutionary algorithm proposed in [20, 21, 22, 23] by professor Ivan Zelinka at the University of Tomas Bata, Zlin. SOMA is stochastic optimization algorithm that is modeled on the social behavior of cooperating individuals [23, 24]. The original idea which led to its creation, is to simulate the behavior of a group of intelligent individuals, who work together to solve a common problem such as finding food sources. SOMA is its robustness in terms of finding the global extremum and its performance is comparable to algorithms such as DE.

SOMA works on a population of candidate solutions (individuals) in loops called migration loops. In each loop, the population is evaluated, and the solution with the highest fitness becomes the leader. Apart from the leader, in one migration loop, all individuals will traverse the input space in the direction of the leader. An individual will travel a certain distance (called the *PathLength*) towards the leader in $n$ steps of defined length. If the path length is chosen to be greater than one, then the individual will overshoot the leader [25]. SOMA's self-organizing progression is the result of mutual interference among the population members, as they determine the direction they will take when searching for a better solution in the area.

### 5.2.1 Parameters of Self Organizing Migrating Algorithm

- $PathLength$ determines the distance from the leading member at which an active individual will stop.

- $Step$ determines the step size of an active individual who is on its way to the leading member.
- $PRT$ is one of the most important and the most sensitive parameters, and it denotes perturbation. The perturbation vector (PRTVector), which describes the direction in which an active individual will be moving, is generated based on this quantity.
- $D$ - The Dimension of a problem, the number of parameters (arguments) of the object function that the algorithm aims to optimize to the best values possible in light of the object function's value.
- $PopSize$ determines the number of individuals present in a population that take part in the migration cycles with the goal of finding the global extreme.
- $The\ Migrations$ gives the number of migration cycles in which reorganization—a rebirth of the whole population aimed at finding fitter individuals occurs.

| Control variables | Recommended interval | Remark |
|---|---|---|
| $PathLength$ | $< 1.1; 5 >$ | Controlling parameter |
| $Step\ (t)$ | $< 1.1; PathLength >$ | Controlling parameter |
| $PRT$ | $< 0; 1 >$ | Controlling parameter |
| $D$ | Problem-specific | Number of arguments in fitness function |
| $PopSize$ | $< 10; Userdefined >$ | Controlling parameter |
| $Migrations$ | $< 10; Userdefined >$ | Stopping parameter |

Table 2: Operating parameters for SOMA [14]

### 5.2.2 Self Organizing Migrating Algorithm Strategies

- *The AllToAll* - lacks a leading individual. Population members thus do not migrate towards the fittest member; rather they move towards all other members. If they find a better extreme on their way, they will complete their journey towards the other members, and only after they have done so they will return to the position at which the best extreme was found. It is at this position that they start at in the next migration cycle [14].
- *AllToAllAdaptive* - is the same strategy as the *AllToAll*. The difference is at the point of return to the better found position. If members find a better extreme in their locomotion, they will complete their migration to the one member they are just migrating to. After that, they will return to the position at which the better extreme was found, and they begin their migration cycle there [14].
- *The AllToOne* - is characteristic for having a specific leading individual (a leader) towards whom the other population members migrate. Once the initial population

has been created, each individual is evaluated by the object function, and its value will determine the future leading individual (the leader) of the next migration cycle. Other population members will then start moving in steps towards the leader [14].

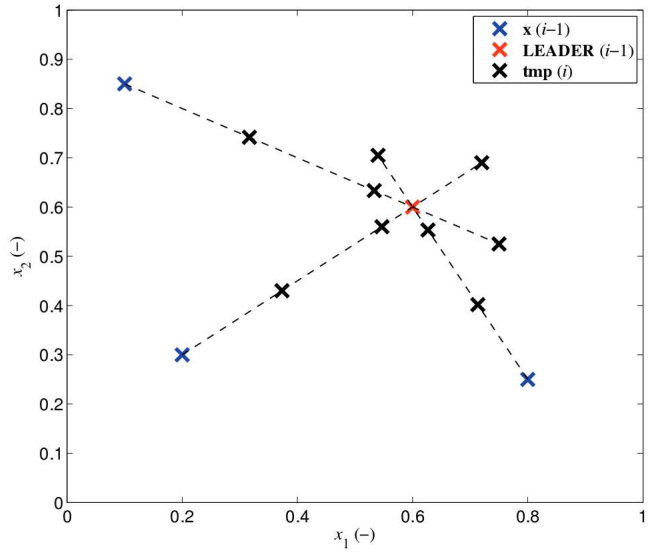



Figure 5: Movement of individuals in the migration loop (*AllToOne* variant,$Step = 3$, $PathLength = 1.3$) [26]

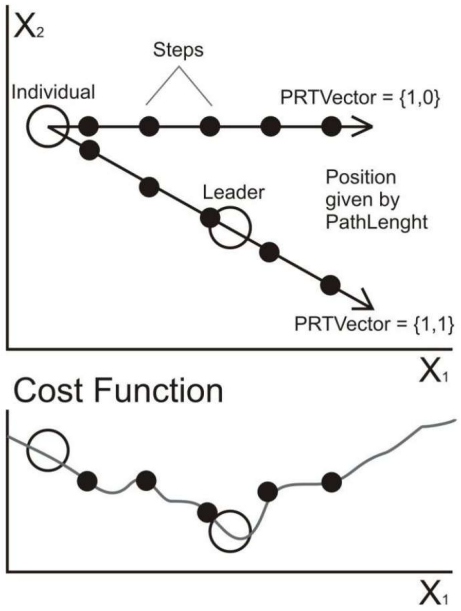



Figure 6: PRTVector and its action on individual movement [27]

- *AllToOneRand* - is based on the principle of individuals migrating towards a leader, where such a leader is not the individual with the best value based on the object function, but rather it is some randomly chosen member of the whole population [14].
- *Clusters* - This version of SOMA with Clusters can be used in any of above strategies. The word *Cluster* refers to calculated clusters. Each individual from the population is tested for the cluster to which it belongs. Each cluster executes SOMA algorithm. Some clusters may be created or anihilated during migration loop [23].

### 5.2.3 The Sequence of Self Organizing Migrating Algorithm's Steps

Because SOMA uses the philosophy of competition and cooperation, the variants of SOMA are called strategies. They differ in the way as to how the individuals affect all others. The best operating strategy is called *AllToAll* and consists of the following steps:

1. *Definition of parameters* - Before execution, the SOMA parameters ($PathLength$, $Step$, $PRT$, $D$, $PopSize$, $Migrations$ see Table 2) are defined.
2. *Creating of population* - The population SP is created and subdivided into clusters.
3. *Migration loop*
   - Each individual is evaluated by the cost function.
   - For each individual the PRT Vector is created.
   - All individuals, perform their run towards the randomly selected according to (3.1). Each solution is selected piecewise. The movement consists of jumps determined by the Step parameter until the individual reaches the final position given by the PathLength parameter. For each step, the cost function for the actual position is evaluated and the best value is saved. Then, the individual returns to the position, where it found the best-cost value on its trajectory.

Mutation, the random perturbation of individuals, is applied differently in SOMA compared with other evolutionary strategies. SOMA uses a parameter called $PRT$ to achieve perturbation. It is defined in the range [0, 1] and is used to create a perturbation vector. As show below:

$$if\ rnd_j \leq\ PRT\ then\ PRTVector_j = 1$$

$$else\ 0,\ j = 1, \ldots, n$$

In standard evolutionary strategies, the crossover operator usually creates new individuals based on information from the previous generation. Geometrically speaking, new positions are selected from an N dimensional hyper-plane. In SOMA, which is based on the simulation of cooperative behaviour of intelligent beings, sequences of new positions in the N-dimensional hyperplane are generated [28].

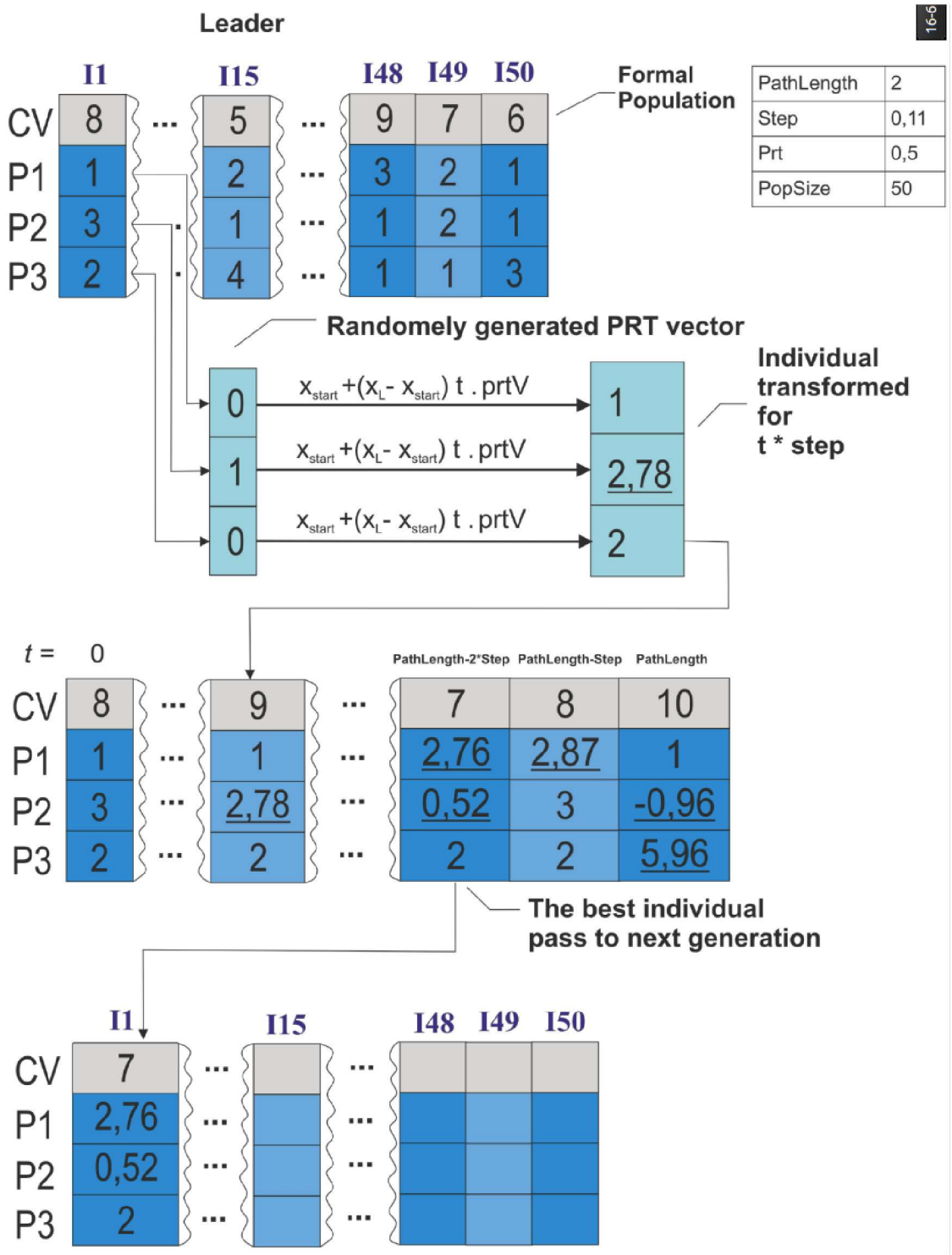



Figure 7: Main parts of SOMA algorithm [27]

# 6 Symbolic Regression

Symbolic regression (i.e., function identification) involves finding a mathematical expression, in symbolic form, that provides a good, best, or perfect fit between a given finite sampling of values of the independent variables and the associated values of the dependent variables. That is, symbolic regression involves finding a model that fits a given sample of data [29].

In past it was only human domain, nowadays there are some methods of symbolic regression more or less effective to find the best mathematical formula. There are three well known methods of symbolic regression - genetic programming, grammatical evolution and analytic programming.

Idea of solving different problems using symbolic regression was first time presented by John Koza, who was using genetic algorithms for genetic programming. This method was improved and verify by years and we can apply today this method to develop very sophisticated electrical circuits.

Grammatical evolution was developed in 90. years of 20. century by Conor Ryan. We can say that the grammatical evolution is improved version of the genetic programming. It's caused due to these algorithms has the same main idea of design programs, but genetic programming is focused on LISP language, while grammatical evolution allows to design programs in any language.

The next method is analytical programming, which I use to solve assigned problems. This algorithm was first time presented by Ivan Zelinka and it differs from grammatical evolution and genetic programming in many ways. Analytical programming can be used with any evolutionary algorithm.

## 6.1 Genetic Programming

Genetic programming (GP) was introduced at late 80. years of 20. century by John Koza. He modified genetic algorithms, where new population is not reproduced with numeric approach, but it is at symbolic level. This means, that new population does not contain numbers, but it contains symbols(mathematical functions, programs, etc.). As an example, we can imagine, that general function set contains mathematical operators$(+, -, *, /, \ldots)$, constants $(x, \pi, \ldots)$ and from these objects we can get expression like $\frac{(x-4)*(sin(x)+1)}{(\pi-5)}$. Set of used functions is in canonical version of GP formed only from function from LISP language.

By evolution steps, we may register effect called Bloat. It's when sequence by children if longer than sequence by parents. This sequence length usually grows linear.

### 6.1.1 Expression in Genetic Programming

In genetic algorithms is every parameter of individual called gene. For simple view canonical version of GP uses tree structure. The top most node in tree is called root. We get result expression reading the tree from bottom to the root.

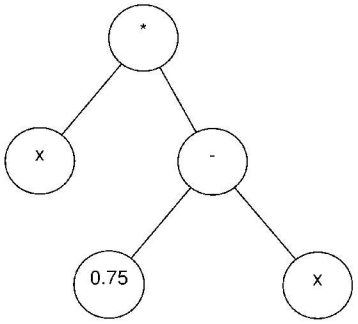



Figure 8: Tree structure of $x * (0.75 - x)$ function

### 6.1.2 Crossover in Genetic Programming

Crossover is phase, when we choose the point where we split and change the trees between them and we create new children.

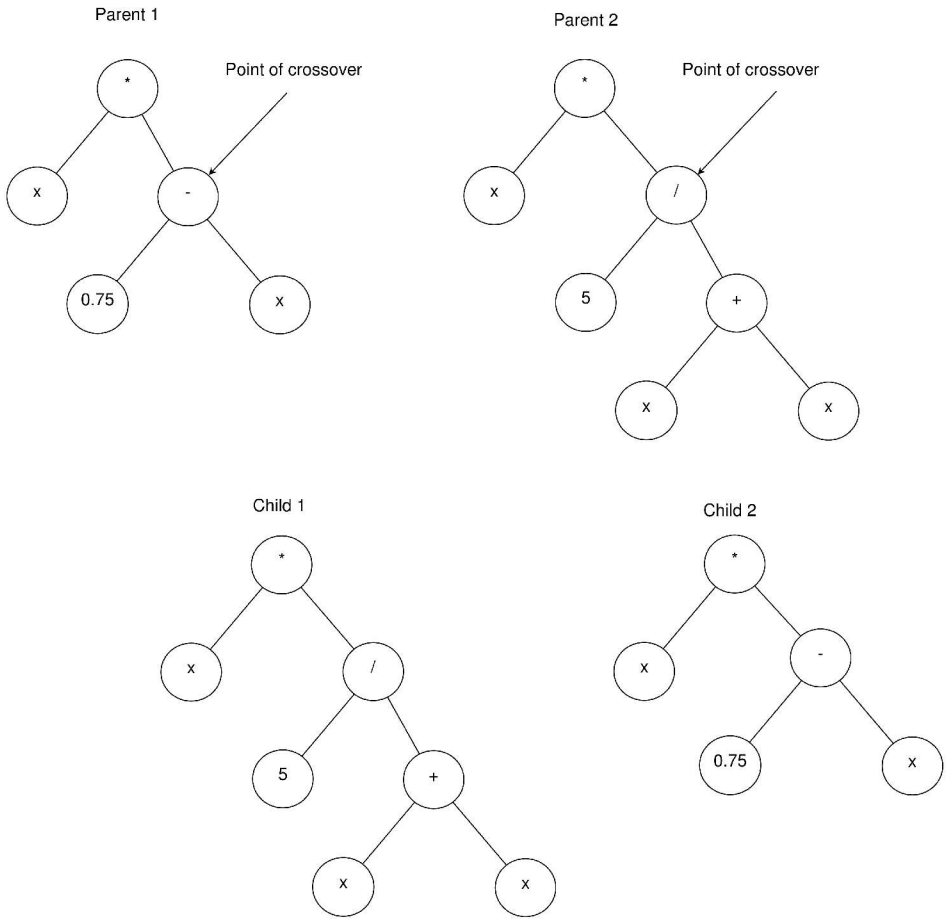



Figure 9: Crossover in GP. Result functions are $\frac{5x}{x+x}$ and $x * (0.75 - x)$

### 6.1.3 Mutation in Genetic Programming

Mutation phase is similar to crossover. We choose the point in tree and its subtree we replace with random generated sequence, or structure. Or we can simply change an information present in node.

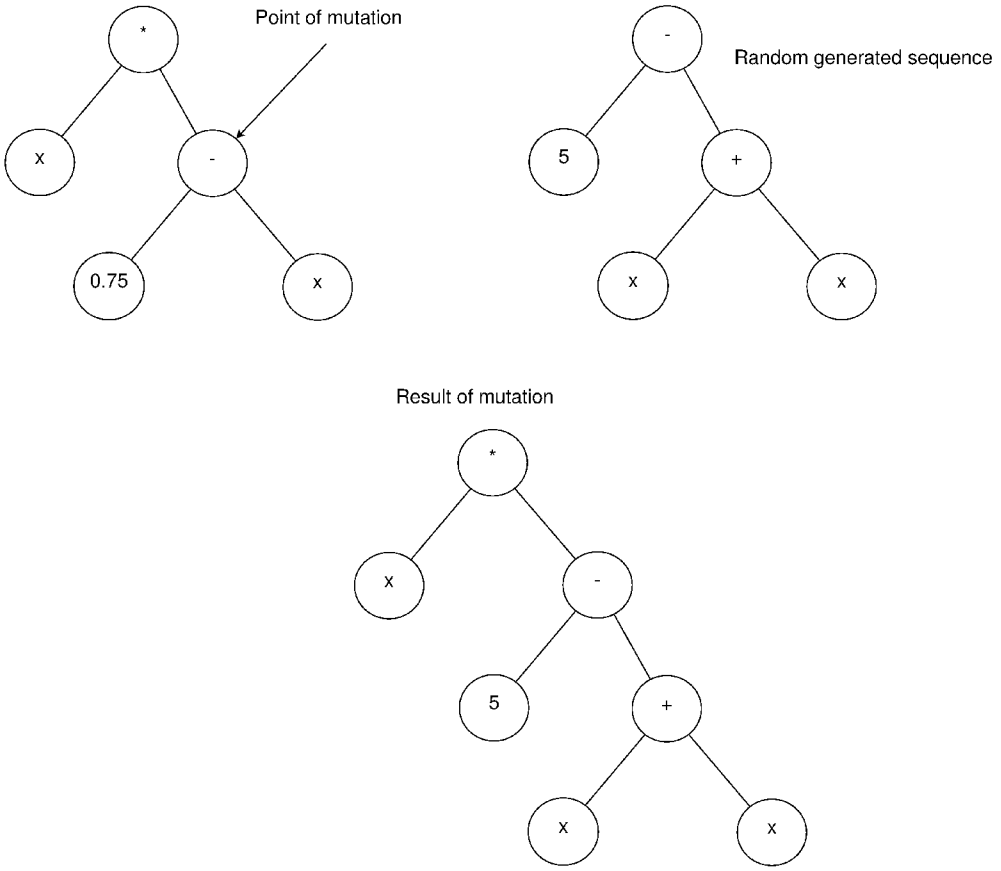



Figure 10: Mutation in GP. Result function is $x * (5 - 2x)$

## 6.2 Grammatical Evolution

The next method in symbolic regression is grammatical evolution (GE). This method was proposed by Ryan, Collins and O'Neill in 1998 [30]. GE has big advantage, that it can design programs in every language, that can be describe with context-free grammar in Backus–Naur Form (BNF), against of GP, where GP was originally written in LISP language. It differs from GP mainly using linear genome, the binary genome determines which production rules in a BNF grammar definition is used in a genotype-to-phenotype (program) mapping process to a program [31]. Identity are represented by rules of grammar. The rules are coded into binary string and we are using one-point crossover. This representation of individual takes less space.

### 6.2.1 Backus–Naur Form

The BNF (Backus Normal Form or Backus–Naur Form) with the van Wijngaarden form are two main notation techniques for context-free grammars. We can define a grammar $G = \{N, T, P, S\}$, where

- $N$ is a finite set of nonterminal symbols. Nonterminal symbols are those symbols which can be replaced with sequence of terminals or nonterminals.
- $T$ is a finite set of terminals, disjoint from $N$
- $P$ is a finite relation from $N$ to $(N \cup T)^*$
- $S$ is a start symbol

In a BNF is a set of derivation rules, written as

$$< symbol > ::= \_\_expression\_\_$$

where `<symbol>` is a nonterminal, and the `__expression__` is sequence terminal and nonterminal symbols. We apply this rules, until we don't get a sequence of only terminal symbols.

### 6.2.2 Encoding of Individual in Grammatical Evolution

In canonical version of GE is 8 bits long binary subsequence of individual labeled as codon. Complete sequence of codons - individual is labeled as chromosome. Chromosomes can have variable length. They define sequence of derivation rules, which we will apply. From codons we extract integer values and we apply rule at position, we will get from following this equation:

$$rule = integer\_value\_of\_codon \; MOD \; count\_of\_rules\_for\_target\_nonterminal$$

We begin to replace nonterminals in the expression from left to right. It can lead in two different states:

- all nonterminals were replaced to terminals
- some nonterminals were not replaced and we reached end of the chromosome

In first case, when all nonterminals are replaced to terminals, we finished at this point replacing. In second case, when we reach the end of the chromosome and some nonterminals were not replaced, we begin read chromosome again from the beginning. It might happen, that we create infinite loop. To prevent this situation we can introduce limit to loop iterations.

Following example shows us individual steps by creation of simple program using GE.

**Example 6.1**
For first, we have to define nonterminals $N$, subsequently terminals $T$, next is start symbol $S$ and at the end we define derivation rules. The definition of BNF is written as:

$$N = \{expr, op, pre - op, var\}$$

$$T = \{cos, +, -, *, /, x, 0.75\}$$

$$S = expr$$

Definitions of substitution rules for result of rule equation mentioned above. Item numbers represent the remainder from division.

- <expr>
  0. <expr><op><expr>
  1. <pre-op> (<expr>)
  2. <var>
- <op>
  0. $+$
  1. $-$
  2. $*$
  3. $/$
- <pre-op>
  0. $cos$
- <var>
  0. $x$
  1. $0.75$

| 00011110 | 00101001 | 00000100 | 01111010 | 10010111 | 01010111 |
|---|---|---|---|---|---|
| 30 | 41 | 4 | 122 | 151 | 87 |
| 11010100 | 01011010 | 00101100 | 10110110 | 01000111 | |
| 212 | 90 | 44 | 182 | 71 | |

Table 3: Sample chromosome

Building of expressions proceeds Cílem této diplomové práce je diskutovat o způsobech užitečných pro analýzu spekter pomocí analytického programování a jeho rychlé realizace. Mým cílem je vytvořit matematické vzorce emisních čar ze spekter, která jsou charakteristická pro Be hvězdy. Jedním z možných řešení tohoto úkolu je symbolická regrese, která v naší aplikaci představuje proces, kdy naměřená data jsou namodelované na

nejlépe reprezentující matematický vzorec. V současné době existují počítačové metody, které nám umožňují provádět tyto výpočty více či méně efektivně. Novou metodou symbolické regrese, ve srovnání s genetickým programováním a gramatickou evolucí, je analytické programování. Cílem této práce je ověřit účinnost paralelního přístupu tohoto algoritmu, pomocí CUDA architektury, který lze spustit na serveru. Dále se budu zabývat realizací náhodných rozhodovacích lesů klasifikováním obrovských množství různých spekter s pomocí matematických funkcí získaných pomocí analytického programování, jak je uvedeno v malém příkladu.as follows:

1. We begin with start symbol <**expr**>
2. 30 mod 3 = 0 - **expr** is replaced given rule <**expr**><**op**><**expr**>
3. 41 mod 3 = 2 - <**var**><**op**><**expr**>
4. 4 mod 2 = 0 - $x$ <**op**><**expr**>
5. 122 mod 2 = 0 - $x*$ <**expr**>
6. 151 mod 3 = 1 - $x*$ <**pre-op**> (<**expr**>)
7. to replace pre-op we don't use codon - $x * cos(<\textbf{expr}>)$
8. 87 mod 3 = 0 - $x * cos(<\textbf{expr}><\textbf{op}><\textbf{expr}>)$
9. 212 mod 3 = 2 - $x * cos(<\textbf{var}><\textbf{op}><\textbf{expr}>)$
10. 90 mod 2 = 0 - $x * cos(x$ <**op**><**expr**>)
11. 44 mod 4 = 0 - $x * cos(x+$ <**expr**>)
12. 182 mod 3 = 2 - $x * cos(x+$ <**var**>)
13. 71 mod 2 = 1 - **result expression is**: $x * cos(x + 0.75)$

■

### 6.2.3 Crossover in Grammatical Evolution

We are generally using in GE one-point crossover. In this crossover we split chromosome in two parts as we can see at Figure 11. Effect of crossover is bigger as in GP, where we change only subtrees.

## 6.3 Analytical Programming

Analytical programming (AP) was proposed in 2005 [32]. This method precede GP and GE.

The term analytic programming was coined by the authors of this article as a matter of simplicity: Because it is possible to use almost any evolutionary algorithm (also experimentally tested) for AP, each EA used for the new approach would add its name to the

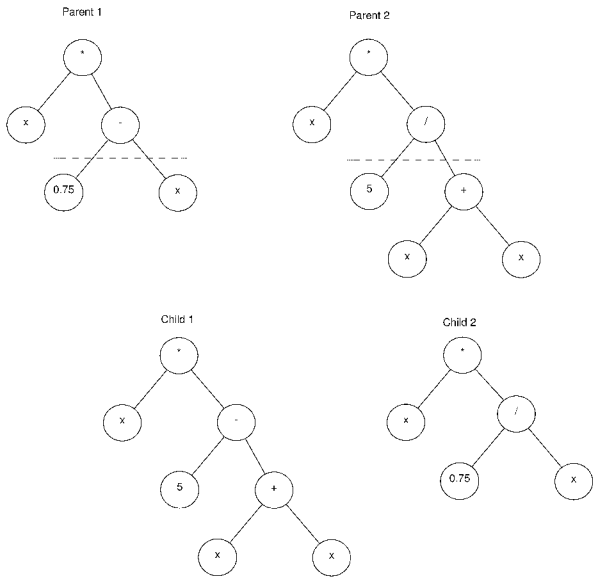



Figure 11: Tree structure of $x * (0.75 - x)$ function

emerging algorithm, e.g. SOMA programming, DE programming, SA programming etc. This clearly would be confusing and complicated. Analytic programming indicates the use of an EA for analytic solutions synthesis (i.e. symbolic regression), thus it was the main reason for choosing the term 'analytic programming' [32].

### 6.3.1 Hilbert Functional Spaces and General Function Space

Analytic programming was inspired by the numerical methods in Hilbert functional spaces and by GP. The principles of AP are somewhere between these two philosophies: From GP stems the idea of the evolutionary creation of symbolic solutions, whereas the general ideas of functional spaces and the building of resulting function by means of a search process (usually done by numerical methods such as the Ritz or Galerkin method) are adopted from Hilbert spaces[32].

A Hilbert space is an abstract vector space possessing the structure of an inner product that allows length and angle to be measured. Hilbert space satisfies condition of orthonormality and orthogonality. Furthermore, Hilbert spaces are complete: there are enough limits in the space to allow the techniques of calculus to be used. For a better understanding, we can imagine Hilbert space as a space whose axes are orthogonal to each other and each axis is typically periodic function (sin, cos,...). Location of the point in this space is described by coordinates of all axes. Composition of all components is giving to us the resulting function.

AP uses specific functional space, which uses a special way to develop the required solutions and it is called general function set. Individual axes of general function set are not orthogonal to each other. While in Hilbert space is represented a certain function

as one axis, in the general function set is each point represented on the axis as a certain function.

### 6.3.2 Discrete Set Handling

The core of AP is based on a set of functions, operators and so-called terminals, which are usually constants or independent variables as well as in GP and GE. The main aim of AP is to synthesize a suitable program which would fit the measured data as well as possible (with the given precision). For this reason, a discrete set handling (DSH) idea [33], [14] was adopted in AP. [34]. Discrete set handling creates an interface between the Evolutionary Algorithm (EA) and the problem. Therefore, we can use in AP almost any evolutionary algorithm. The individual is represented as a non-numerical value and a numerical value is added to the evolutionary process as an integer index. This index represents an individual from the General Function Set.

### 6.3.3 Versions of Analytical Programming

There are three versions of AP. $\text{AP}_{basic}$ is a basic version of AP and uses constants from the terminal set. $\text{AP}_{meta}$ - in this version there are constants not defined in the terminal set; in the terminal set there is only one general constant $K$ and every constant $K$ is estimated by a different or the same evolutionary algorithm. By synthesis of program, are constants indexed as $K_1, K_2, K_3, ..$ and than are estimated. $\text{AP}_{meta}$ get its name, because it is running metaevolution by estimating constants. We can mention one disadvantage of this version – because of running evolution under evolution, it could be very slow for a large number of steps. There is a big number of evaluations of the fitness function. The last version is $\text{AP}_{nf}$ – constants are estimated by non-linear fitting algorithm. This method gives stable performance by estimating constants.

### 6.3.4 General Function Set

The General Function Set (GFS) is a set of all mathematical objects - functions, operators and terminals (usually constants, variables,. . . ).

- operators $+, -, *, /, \wedge, \vee, \ldots$
- functions $sin, cos, tan, log, \ldots$
- terminals $\pi, x, y, 0.213, 1.546, \ldots$

GFS may consists of functions with different numbers of input arguments. From all these object is synthesized a solution. GFS is user-defined, so the content may differ. We must split the content of GFS into classes based on the numbers of arguments:

$$GFS_{all} = \{+, -, *, /, mod, sin, tan, t, x, \pi, \ldots\}$$

$$GFS_{0args} = \{t, x, \pi\, 0.75, 5, \ldots\}$$

$$GFS_{1args} = \{sin, cos, tan, abs, \ldots\}$$

$$GFS_{2args} = \{+, -, *, /, mod, \ldots\}$$

$$GFS_{3args} = \{User\_Function, LerchPhi, \ldots\}$$

Choosing the right set may have a large influence on the convergence of AP. This division into classes ordered by number of arguments is necessary for avoiding creating of pathological identity, I describe later.

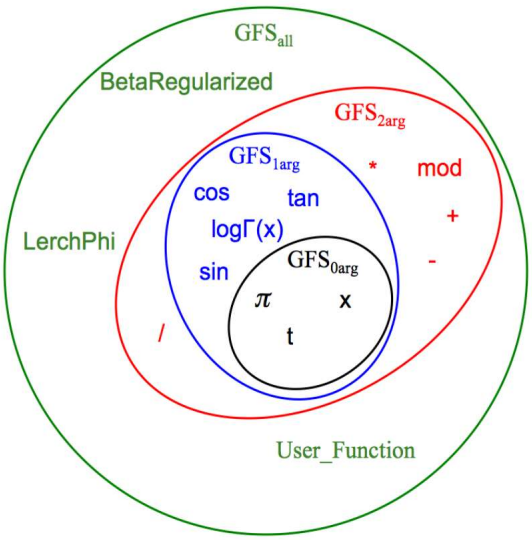



Figure 12: General Function Set example

### 6.3.5 Evolutionary Algorithm

AP was designed to be a very robust method and we can use almost any evolutionary algorithm. Individual steps, such as mutation, crossover, etc. are fully handled by the chosen evolutionary algorithm. Operations which make EAs when the algorithm is tun do not have any influence on performance. Thus the result performance depends mainly on the correct choice of GFS individuals. For every evolutionary algorithm, the main goal is generally to reduce the fitness value to below a user-defined threshold, or to achieve maximum number of migrations for a Self Organizing Migrating Algorithm (SOMA), or in the case of Differential Evolution (DE) - generations.

### 6.3.6 Mapping Operators

Mapping is the phase when an individual is transformed into a useful mathematical function. It consist of two parts, DSH and security functions, to exclude the creation of pathological individuals. In the evolutionary algorithm the individual is represented by a vector of indexes and is remapped to mathematical objects from GFS.

### 6.3.7 Reinforced Evolution

By running evolution, more or less suitable individuals are created, so one very good idea is to include the best individual as the terminal in GFS. The main idea of reinforcement is based on the addition of a just-synthesized and partly successful program in an initial set of terminals [35]. The decision on whether the best value will be added to GFS is ensured by a user-defined threshold value. If it is reached, from that moment it is added in GFS as a terminal, best solution and updated whenever a better solution with lower fitness is found.

### 6.3.8 Security Procedures

Evolution can result in an expression, which is not mathematically correct, thus we create a pathological individual(without argument). We can prevent this by distributing GFS into classes ordered by the number of arguments. By mapping from evolution space to GFS, we measure the distance to the end of the expression. When the number of arguments is greater than the distance to the end of the expression, we choose individuals from the lower class. We must also pay attention so as to exclude errors from the fitness function, such as division by zero, functions with an imaginary or real part (if not expected), frozen functions (an extremely long time to get a cost value), etc. [35]

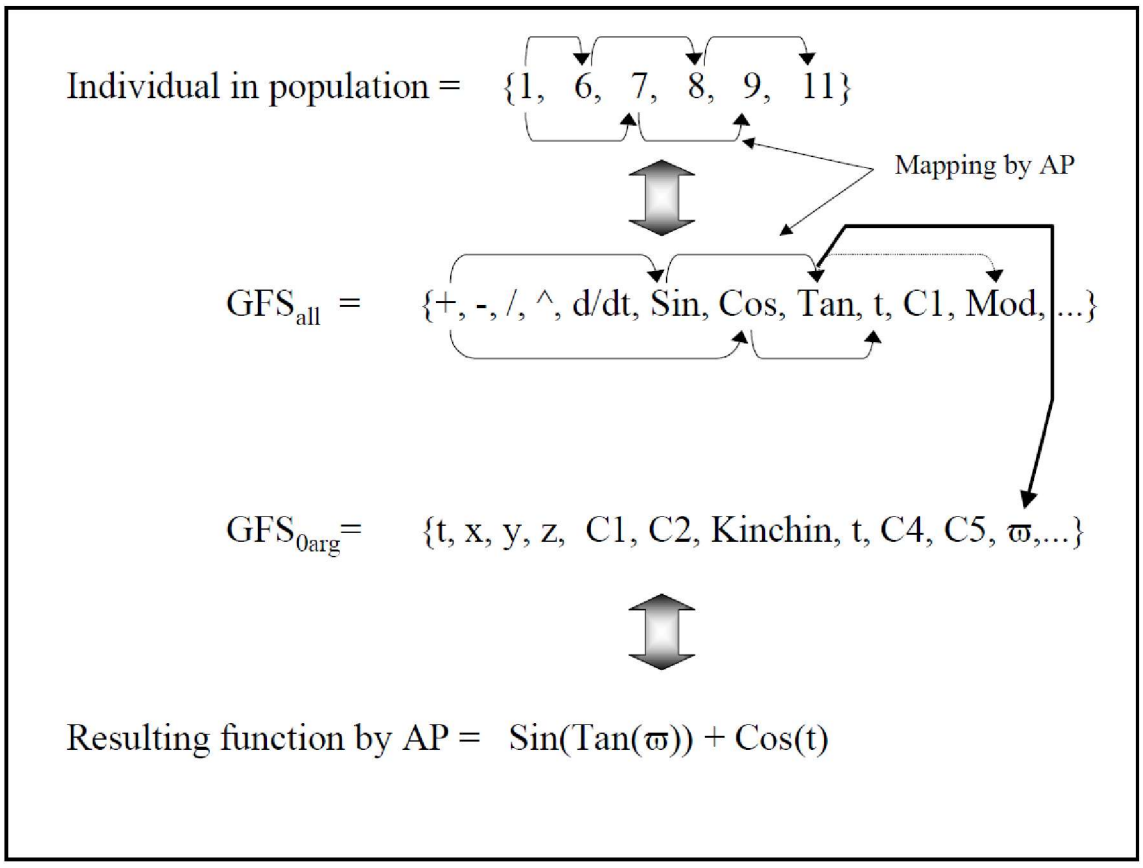



Figure 13: Schema of mapping and security principles. Because of measuring distance to end of expression is $tan$ replaced by $\omega$ [32]

# 7 Parallel Implementation using CUDA - an Overview

## 7.1 Compute Unified Device Architecture

Compute Unified Device Architecture (CUDA) is propriety parallel computing architecture developed by Nvidia Corporation as an API extension of the C programming language [36], [37], [38], which allows the definition of specific functions from a normal C program to run on the GPU's stream processors. CUDA was released in November 2006, with new generation of graphic cards GeForce 8000 series.

Compared to the central processing unit (CPU) that contains one or several sophisticated processors working at high clock speed, graphics processing unit (GPU) consists of hundreds of SPs having simplified structures and working at lower clock speed.

The CUDA API (both low level and high level) provides a platform for accessing the Nvidia GPU for processing. This allowed a programmer to bypass the traditional OpenGL or Direct3D techniques which were needed to program the chips. Additionally, and most importantly, C language can now be used to program for CUDA, through the PathScale Open64 C compiler. This has essentially introduced CUDA for mainstream programmers [39].

CUDA also refers to an intuitive and scalable programming model based on an extended C programming language, called CUDA-C. CUDA-C contains three types of functions:

1. *host functions* - called,executed on host. These functions are exactly the same functions available in C language.

2. *kernel functions* - called on host,executed on device. It requires a qualifier `__global__` being declared before the function's return value type that must be void

3. *device functions* - called,executed on device. It requires a quantifier `__device__` being declared before the function's return value type that can be any types.

CUDA is SIMT (Single Instruction Multiple Threads). Its main feature is that one instruction is executed by thousands of threads. All cores in the same group execute the same instruction at the same time, much like classical SIMD processors. SIMT handles conditionals somewhat differently than SIMD, though the effect is much the same, where some cores are disabled for conditional operations. However, kernels can effectively perform only basic operations.

The general outline of CUDA is given in 16. The CPU is able to communicate with the GPU using the PCIe bus, and therefore the communication speed is limited by the bus speed. A GPU itself has a number of attributes. The GPU is made up of a number of blocks. Each block is subsequently divided into a number of threads. Each block has its own shared memory and registers, which can be accessed by all the threads residing in that particular block. All blocks can access the global memory and the shared memory of the GPU. The minimum applicable amount of blocks available is 65,535 and each block has 512 threads each. CUDA processing cores are referred to as kernels, which

are launched from the CPU. Kernels can be made up of any combinations of blocks and threads, depending on the application [37].

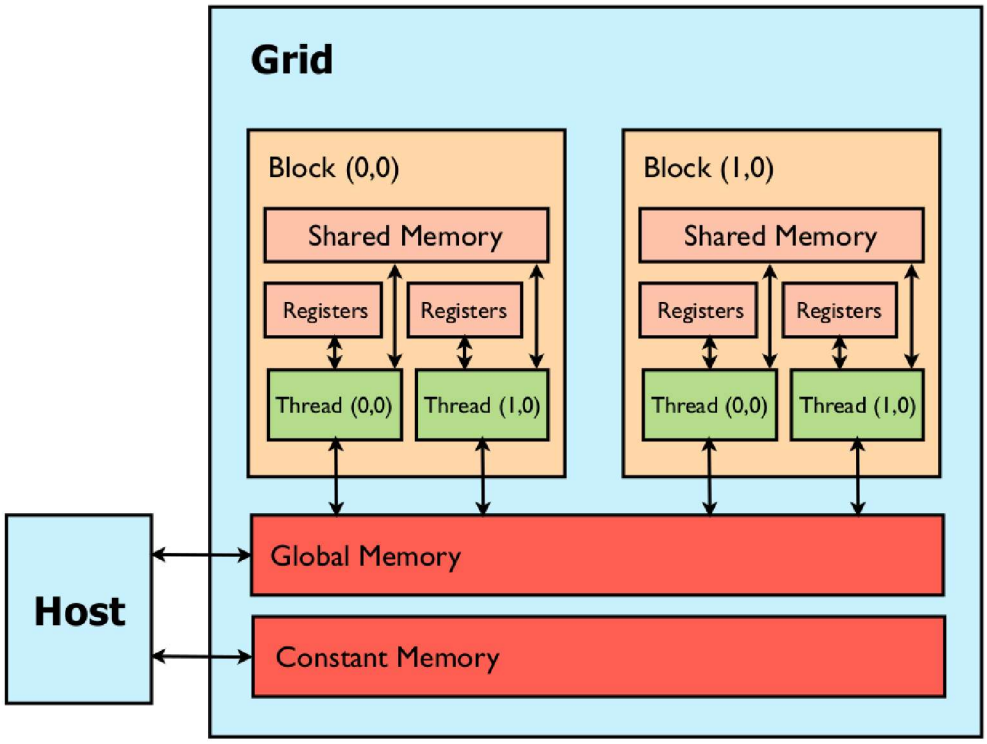



Figure 14: CUDA outline[37]

When launching a kernel function, its associated kernel execution configuration parameters, such as *gridDim* and *blockDim*, must be specified within <<<...>>>. When determining configuration parameters, we should consider: each thread block has a maximal number of threads determined by the device's compute capability; each SM has the limited shared memory size and register number, which will influence the allowed number of threads per block and the allowed number of blocks per SM; all threads in one block can access the same data stored on shared memory while threads in different blocks can only communicate via global memory [40].

In the CUDA device various types of memory reside, as you can see in Table 4. They differ in many ways and right choosing of memory, which use our program, may have a key influence to performance of implemented program. Individual types of memory are described below [38]:

- *Registers* - are the fastest memory on the GPU. They are a very precious resource because they are the only memory on the GPU with enough bandwidth and a low enough latency to deliver peak performance. Each GF100 SM supports 32 K 32-bit

registers. The maximum number of registers that can be used by a CUDA kernel is 63, due to the limited number of bits available for indexing into the register store.

- *Local memory* - Local memory accesses occur for only some automatic variables. An automatic variable is declared in the device code without any of the `__device__`, `__shared__`, or `__constant__` qualifiers.

- *Shared Memory* - Shared memory (also referred to as smem) can be either 16 KB or 48 KB per SM arranged in 32 banks that are 32 bits wide. Contrary to early NVIDIA documentation, shared memory is not as fast as register memory.

- *Constant Memory* - For compute 1.x devices, constant memory is an excellent way to store and broadcast read-only data to all the threads on the GPU. The constant cache is limited to 64 KB. It can broadcast 32-bits per warp per two clocks per multiprocessor and should be used when all the threads in a warp read the same address. Otherwise, the accesses will serialize on compute 1.x devices.CUDA Memory Types Compute 2.0 and higher devices allow developers to access global memory with the efficiency of constant memory when the compiler can recognize and use the LDU instruction.

- *Texture Memory* - Textures are bound to global memory and can provide both cache and some limited, 9-bit processing capabilities. How the global memory that the texture binds to is allocated dictates some of the capabilities the texture can provide. For this reason, it is important to distinguish between three memory types that can be bound to a texture.

| **Memory** | **Location** | **Cached** | Access | **Scope** |
|---|---|---|---|---|
| Register | On-chip | No | Read/Write | One thread |
| Local | On-chip | Yes | Read/Write | One thread |
| Shared | On-chip | N/A | Read/Write | All threads in a block |
| Global | Off-chip (unless cached) | Yes | Read/Write | All threads + host |
| Constant | Off-chip (unless cached) | Yes | Read | All threads + host |
| Texture | Off-chip (unless cached) | Yes | Read/Write | All threads + host |

Table 4: CUDA Memory Types and Characteristics [38]

## 7.2 Differential Evolution on the GPU

DE can be subdivided into six kernels, described below [40]:

- *kernel(I)* - initializes the population and writes it into global memory. This kernel requires random numbers.

- *kernel(E)* - evaluates the objective function values of population members as per the problem being solved and writes them into global memory. The evaluation of each

population member can take up one or more threads. This kernel needs to read the population to be evaluated from global memory.

- *kernel(P)* - prepares the mutually exclusive indices of randomly sampled population members for each target vector to generate the mutant vector. The indices are structured into a matrix and written into global memory. This kernel requires random numbers.
- *kernel(M)* - performs the DE mutation to generate mutant vectors, which are then written into global memory. This kernel needs to read the current population from global memory and requires random numbers.
- *kernel(C)* - performs the DE crossover to generate trial vectors, which are written into global memory. This kernel needs to read the current population and mutant vectors generated in *kernel(M)*from global memory and requires random numbers.
- *kernel(R)* - performs the objective function values comparison between a trial vectors and their corresponding target vectors to form the population of the next generation, which is then written into global memory. This kernel needs to read the objective function values of trial and target vectors, trial vectors and the current population from global memory.

## 7.3 Self Organizing Migrating Algorithm on the GPU

There are more different approaches to implement paralel vesrion of SOMA algorithm. We can mention these variants of SOMA *AlltoAll* algorithm:

1. Variant - The CPU is only responsible for memory copies and launching of a number of kernels equal to a population size. Each CUDA thread is responsible for migration of one individual towards all the other individuals in population. This variant puts a lot of load on each thread resulting in long running kernels [41].
2. Variant - This variant shifts a part of load from GPU to the CPU. The CPU loops over all individuals in the population and launches *populationSize* kernels. Each CUDA thread computes migration of one individual towards one target individual and stores the best found solution in a global array. After the kernels finish, another kernel is launched which finds the best individual among the kernels and writes it into temporary population. To speedup the operation, the search for the best individual is implemented as a parallel reduction. After the migration of all individuals ends, the individuals from the temporary population are copied into population and new round can begin [41].
3. Variant - Each kernel computes migration of one individual towards another one, as in Variant 2, but migration of the whole population is not implemented via CPU loop, but by utilizing high number of kernel launches. Each kernel performs migration of an individual towards another individual and writes best found position and fitness value into global memory [41].

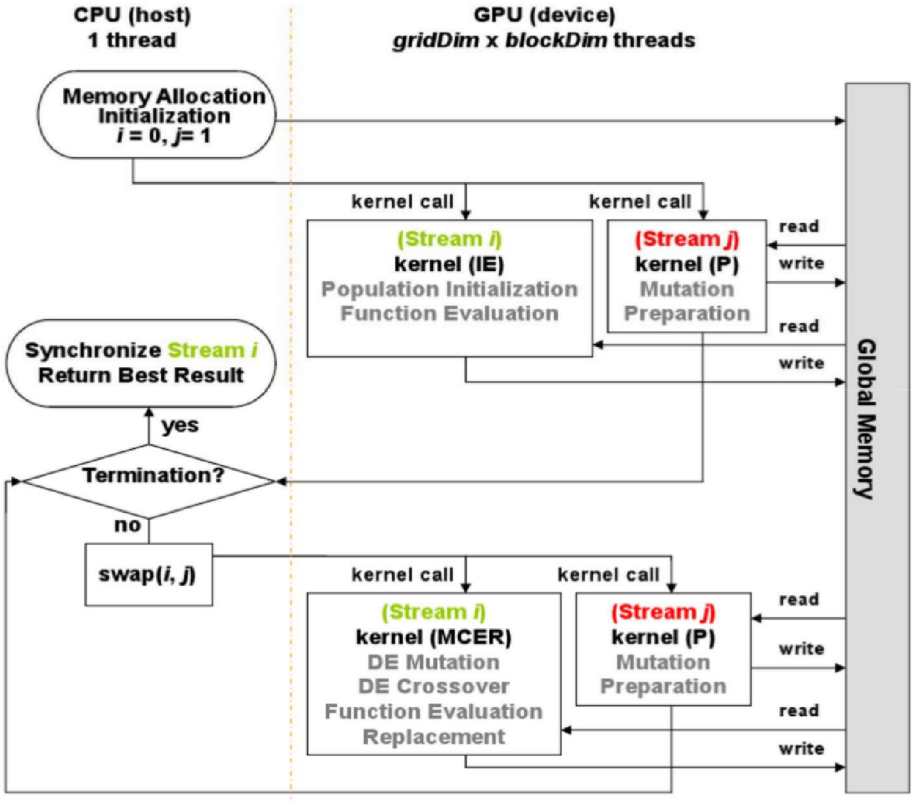



Figure 15: Improved CUDA DE flowchart [40]

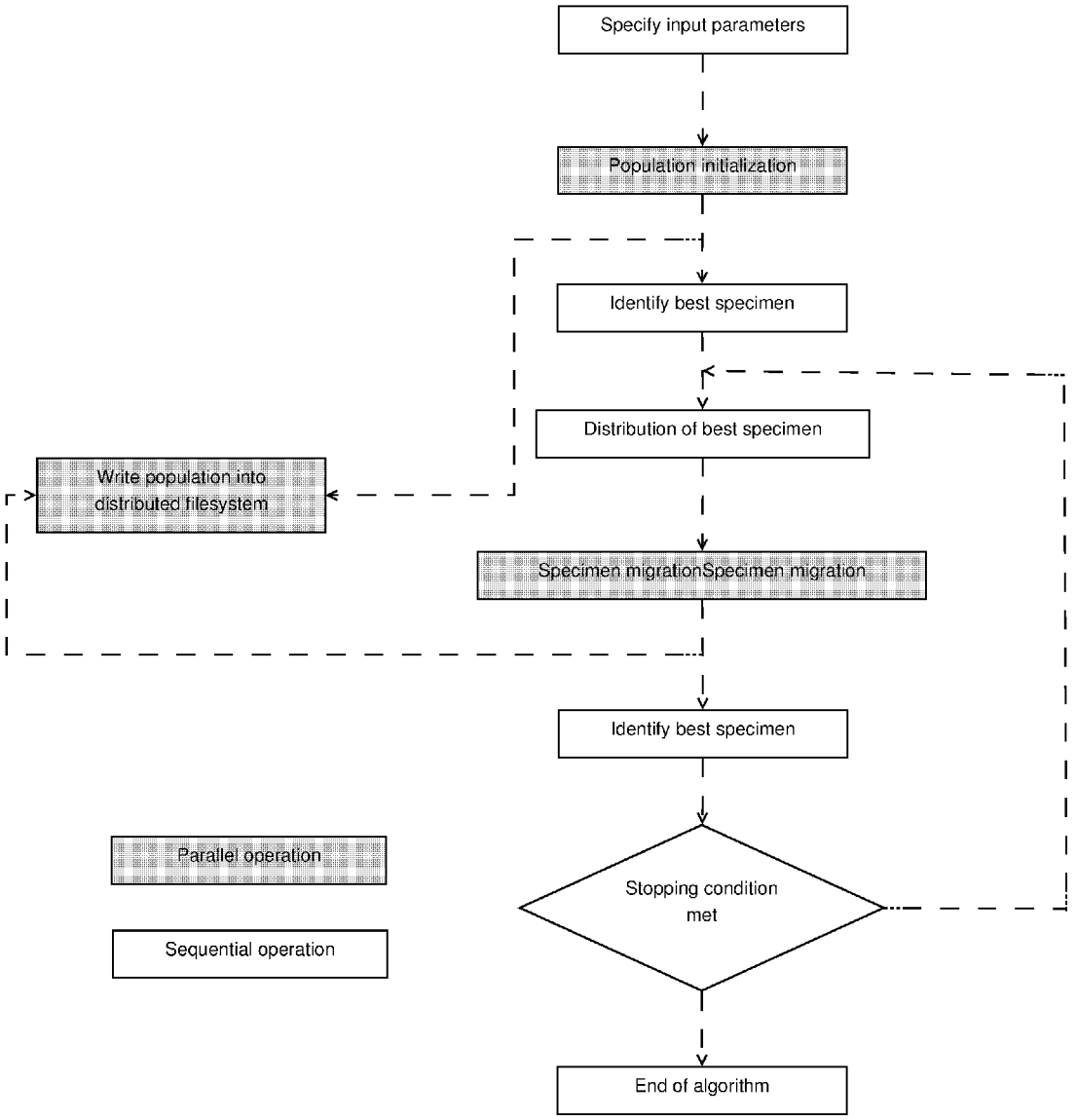



Figure 16: CUDA SOMA implementation approach flowchart

# 8 Parallel Implementation - Methods, Datasets, Results

## 8.1 Parallel Implementation of Analytical Programming

If the program is run on a device which also serves as an output device for GUI, long running kernels may be terminated by system watchdog timer. Therefore the most complex problems should be solved without GUI or another GPU should be present for displaying the GUI, or we can disable this watchdog in windows registry.

The idea of implementing AP on parallel architecture is based on successful parallel implementation of evolutionary algorithms on CUDA, where a significant speeding up was achieved [42], [43]. We were inspired to implement a parallel version of AP on CUDA, because of the good results achieved by the evolutionary algorithms when implemented on CUDA, although execution of this number of operations and structures may lead to worse performance results, as expected.

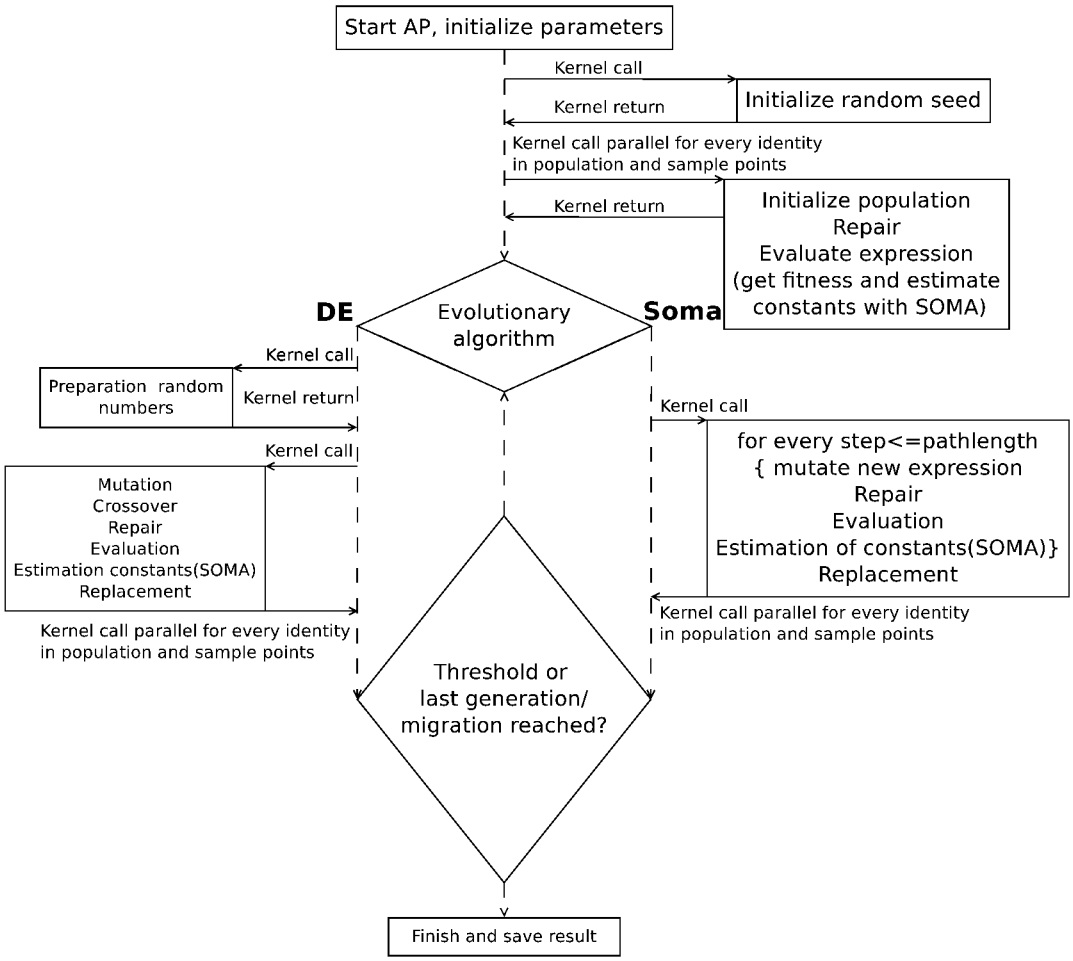



Figure 17: The flowchart of AP implementation

To successful run of my AP implementation, we have to set up variables in *set.h* file and there we are choosing, which algorithm with its settings we want to run.

After initialization phase, where I allocated constants and variables on GPU, I call first kernel `kernelSRC`, which initialize random seed, or initialize values for later deterministic chaos usage. Next I call `kernelIE`, which initialize population and compute values of constants using metaevolution. This part of code is common for both - SOMA and DE main evolution.

```
//running kernels
  kernelSCR<<<1,np>>>(devStates,(unsigned long)time(NULL));
  kernelIE<<<grid1,block,0,stream_1>>>(devStates,population,d_GFS,d_GFS0,expression,
      constant,origx,origy,fitness_ee,dpop_tmp1,dpop_tmp2,leader_ee);
```

Listing 2: Initialzation before main evolution cycles

The code is different for SOMA and different for DE after this phase. The main loops for SOMA and DE are shown in codes below:

```
while(gen_tmp != migration)
  {
    /*Memcpy of variables*/
    kernelSOMA<<<grid1,block>>>(devStates,population,pop_tmp,leader_d, d_GFS,d_GFS0,
        expression,constant,constant_tmp,origx,origy,best,fitness_ee,dpop_tmp1,dpop_tmp2,
        leader_ee);
      SAFE_CALL(cudaMemcpy(pop,population, pop_size*sizeof(float),
          cudaMemcpyDeviceToHost),"CudaMemcpy pop Failed");
      SAFE_CALL(cudaMemcpy(consta,constant, pop_size*cpopsize*sizeof(float),
          cudaMemcpyDeviceToHost),"CudaMemcpy consta Failed");
      getbestvec(tx,np,d,pop,consta);          SAFE_CALL(cudaMemcpy(best,bestvec,tx*sizeof(
          float),cudaMemcpyHostToDevice),"CudaMemcpy origy Failed");
      gen_tmp++;
      if ( fitch != best_fitness )
      {
        fitch = best_fitness;
        for(int i = 0 ; i<=d;i++)leader[i]=pop[(d+1)*pos+i];
      }
      if ( best_fitness < accuracy)
      {
        gen_fmp=migration;
        myfile<<"Treshold reached!"<<endl;
      }else
      {
        if (gen_fmp==migration)myfile<<"Last iteration done!"<<endl;
      }
    }
```

Listing 3: Main loop for AP with SOMA main evolution

We can see that SOMA kernel is on GPU the whole migration loop. Every loop is needed to recalculate *leader* and send it to device memory and to get the best expression because of reinforced evolution to include in GFS set. Subsequently, at the end of every iteration is meeting of ending conditions checked.

```
while(gen_tmp != generations)
        {
    /*Memcpy of variables*/
        SAFE_CALL(cudaMemcpyAsync(copym, mutation,np*3*sizeof(int),
            cudaMemcpyDeviceToHost, stream_2),"CudaMemcpyAsync Failed");
        SAFE_CALL(cudaMemcpyAsync(mutation1, copym,np*3*sizeof(int),
            cudaMemcpyHostToDevice, stream_1),"CudaMemcpyAsync Failed");
        kernelP<<<grid1,block,0,stream_2>>>(devStates,mutation);      kernelMCER<<<
            grid1,block,0,stream_1>>>(devStates,population,pop_tmp,mutation1,d_GFS,
            d_GFS0,expression,constant,constant_tmp,origx,origy,best,fitness_ee,dpop_tmp1,
            dpop_tmp2, leader_ee);
        SAFE_CALL(cudaStreamSynchronize(stream_1),"CudaStreamSynchronize stream_1
            Failed");
        SAFE_CALL(cudaMemcpy(pop,population, pop_size*sizeof(float),
            cudaMemcpyDeviceToHost),"CudaMemcpy pop Failed");
        SAFE_CALL(cudaMemcpy(consta,constant, pop_size*cpopsize*sizeof(float),
            cudaMemcpyDeviceToHost),"CudaMemcpy consta Failed");
        getbestvec(tx,np,d,pop,consta);
        SAFE_CALL(cudaMemcpy(best,bestvec,tx*sizeof(float),cudaMemcpyHostToDevice),"
            CudaMemcpy origy Failed");
        gen_tmp++;
        if ( best_fitness <accuracy)
        {
           gen_tmp=generations;
           myfile<<"Treshold reached!"<<endl;
        }else
        {
           if (gen_tmp==generations)myfile<<"Last iteration done!"<<endl;
        }
    }
```

Listing 4: Main loop for AP with DE main evolution

As we can see in code above, there are two kernels in main loop. First, *kernelP* prepare random values to the DE evolution. Subsequently, *kernelMCER* with DE is called and also is checked at the end of every iteration the meeting of ending conditions. The best expression is of course included every loop in GFS, when better is found. Results of AP are written in *result.txt* file.

## 8.2 Pseudo Random Number Generator and Deterministic Chaos

To be able to carry out a comparison in our tests we used a classical pseudo-random number generator *curand* [44] library and on the other site we tried to compare it to implemented deterministic chaos, with setting $A = 4$, which one-dimensional Logistic equation is as follows:

$$x_{n-1} = Ax_n(1 - x_n)$$

The Logistic equation (logistic map) is a one-dimensional discrete-time example of how complex chaotic behaviour can arise from very simple non-linear dynamical equation [45]. This chaotic system was introduced and popularized by the biologist Robert May [46]. It

was originally introduced as a demographic model as a typical predator – prey relationship. The chaotic behaviour can be observed by varying the parameter $r$. At $r = 3.57$ is the beginning of chaos, at the end of the period-doubling behaviour. At $r > 3.57$ the system exhibits chaotic behaviour [47].

$$x_{n+1} = A\,(1 - x_n)\,x_n$$

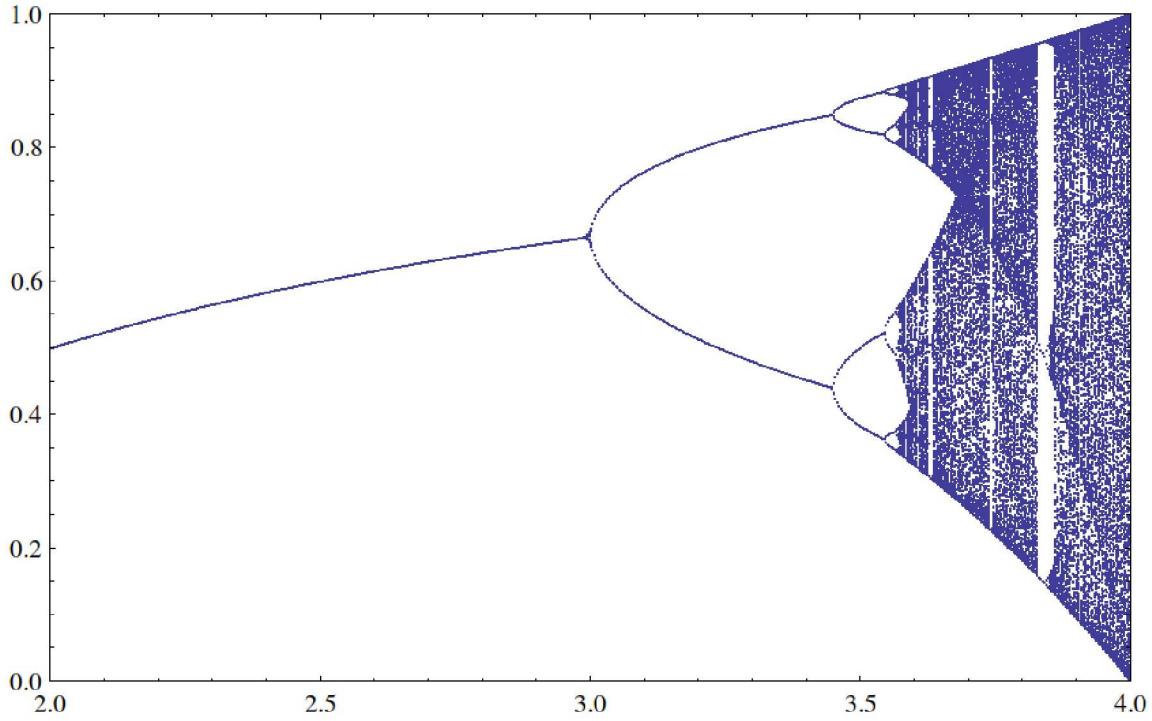



Figure 18: Bifurcation diagram of Logistic Equation. $A =< 2; 4 >$ and number of sample points is $650$.

```
Clear[x];
kvadratic[x_] := a x (1 − x);
a = 4;
KList4 = NestList[kvadratic, 0.2027, 4];
L14 = ListPlot[KList4, Joined −> True, PlotRange −> All,
   DisplayFunction −> Identity];
L24 = ListPlot[KList4, PlotRange −> All,
   PlotStyle −> {RGBColor[1, 0, 0], PointSize[0.02]},
   DisplayFunction −> Identity];
x := x + 0.00001;
Show[L14, L24, DisplayFunction −> $DisplayFunction]
```

Listing 5: Deterministic Chaos implementation in Mathematica

```
__device__ float randFlt(float* globalState, int id,float min, float max)
{
   globalState[id]=A*globalState[id]*(1.f-globalState[id]);
   return min+globalState[id]*(max-min);
}
```

Listing 6: Deterministic Chaos implementation in my CUDA application of Analytical programming

```
__global__ void kernelSCR (float * state, unsigned long seed)
{
    int id= threadIdx.x+blockIdx.x*blockDim.x;
    if (id < NP)
   {
      curandState state1;
      curand_init (seed, id, 0,&state1);
      state[id]= curand_uniform(&state1);
   }
}
```

Listing 7: Initialization random seeds using curand library

The `curand_uniform(&state1)` function returns a sequence of pseudorandom floats uniformly distributed between 0.0 and 1.0. It may return from 0.0 to 1.0, where 1.0 is included and 0.0 is excluded. Distribution functions may use any number of unsigned integer values from a basic generator. The number of values consumed is not guaranteed to be fixed [44].

# 9 Experiments

I carried out tests with SOMA and DE as the main evolutionary algorithm of AP. For estimating constants I chose the SOMA algorithm.

## 9.1 3sine, 4sine, Quintic, Sextic Problems

In this part I will present measured data with this four problems. The settings for our implementation are shown in Table 5.

| DE | | SOMA | | SOMA constants | |
|---|---|---|---|---|---|
| NP | 500 | PopSize | 500 | PopSize | 10 |
| Dimensions | 30 | Dimensions | 20 | Dimensions | 20 |
| Generations | 15 | Migrations | 10 | Migrations | 5 |
| F | 0.9 | PRT | 0.1 | PRT | 0.1 |
| CR | 0.5 | PathLength | 4 | PathLength | 3 |
| | | Step | 0.3 | Step | 0.3 |

Table 5: Algorithm settings

I sampled function with 50 equidistant points in interval $< -1; 1 >$. Constants was generated in interval $< -5; 5 >$. Constant from deterministic chaos was set to $A = 4$. Average values of fitness function from 10 tests are included in Table 6

| Problem | SOMA | DE |
|---|---|---|
| Sextic | 0.0082315 | 0.235671 |
| Quintic | 0.0271185 | 0.449525 |
| 3sine | 8.85088 | 15.46 |
| 4sine | 7.49241 | 17.8147 |

Table 6: Measured average fitness value of 3sine, 4sine, quintic and sextic problems

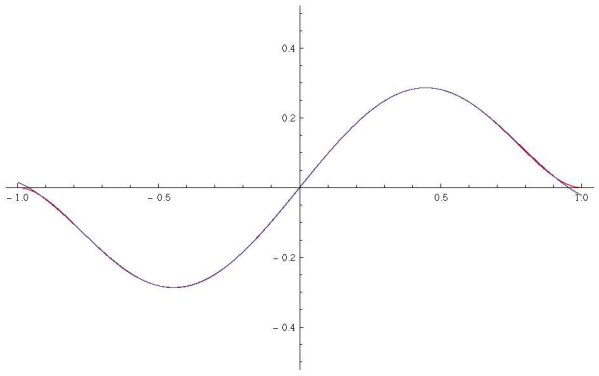



Figure 19: Extracted quintic expression from AP (blue), original quintic expression (red)

## 9.2 Real Spectra

We cut the spectra from Figure 1 and obtained only the emission line in Figure 20 with 50 equidistant points, to satisfy our requirements. The data on the Be stars spectra come from the archive of the Astronomical Institute of the Academy of Sciences of the Czech Republic[10].

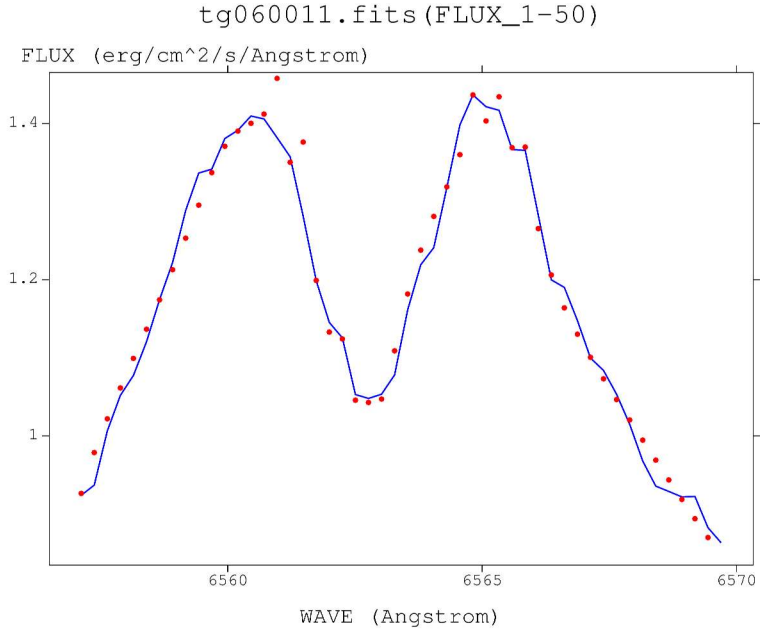



Figure 20: Extracted emission line (blue) of Be star and our best result (red dots) of the SOMA algorithm with deterministic chaos

The settings for our implementation are shown in Table 7.The measured results are shown in Tables 8 and 9.

| DE | | SOMA | | SOMA constants | |
|---|---|---|---|---|---|
| NP | 500 | PopSize | 500 | PopSize | 10 |
| Dimensions | 40 | Dimensions | 20 | Dimensions | 20 |
| Generations | 100 | Migrations | 10 | Migrations | 5 |
| F | 0.9 | PRT | 0.1 | PRT | 0.1 |
| CR | 0.5 | PathLength | 4 | PathLength | 3 |
| | | Step | 0.21 | Step | 0.25 |

Table 7: Algorithm settings

[10] Available from: `http://astropara.projekty.ms.mff.cuni.cz/spectra/newest/`

| | DE PRNG | DE Chaos | SOMA PRNG | SOMA Chaos |
|---|---|---|---|---|
| MIN | 4.02327 | 4.00432 | 0.85327 | 0.83326 |
| AVG | 4.30381 | 4.27877 | 1.41000 | 1.39508 |
| MAX | 4.57460 | 4.51520 | 1.84721 | 1.91367 |

Table 8: Minimum, average and maximum fitness achieved from our tests. Each method was tested 10 times.

| [s] | DE PRNG | DE Chaos | SOMA PRNG | SOMA Chaos |
|---|---|---|---|---|
| MIN | 2688.59 | 1962.53 | 3161.47 | 2761.57 |
| AVG | 2805.84 | 2296.42 | 3305.62 | 2942.40 |
| MAX | 2873.77 | 3037.32 | 3372.46 | 3098.13 |

Table 9: Duration of execution of AP: with SOMA 95238096 evaluation of the cost function in the main EA, with DE 100000000

Thanks to the use of deterministic chaos against a pseudo-random number generator (PRNG) we speed up the running time by using simple mathematical operations, which is good for the CUDA kernel. Convergence of the fitness value is also a bit better.

## 9.3 Classification Results

In Table 10 i tested error rate of tools implementing RDF. Data in `*.arff`[11] file format were obtained from free UCI repository[12]. This data were shuffle. After the data shuffle I split dataset to training (70%) and testing (30%) part.

| [%] | **R** | **Rf-ace** | **Waffles** |
|---|---|---|---|
| ionosphere | 8.6 | 10.4 | 14.3 |
| sonar | 12.7 | 14.3 | 14.3 |
| iris | 4.4 | 2.2 | 2.2 |
| zoo | 9.7 | 10.1 | 6.4 |
| segment | 2.7 | 4.8 | 1.4 |

Table 10: Error rates of RDF tools

Subsequently, I choose dataset *ionosphere.arff*, because it contains spectra data, what satisfies objectives of this master thesis. I modeled function using SOMA algorithm and obtain following function shown at Figure 21:

Subsequently, I insert samples from this function to *ionosphere.arff* and I obtain right classification of this line as *b* type.

[11] ARFF format definition: `http://www.cs.waikato.ac.nz/ml/weka/arff.html`

[12] Free datasets available at: `http://archive.ics.uci.edu/ml/`

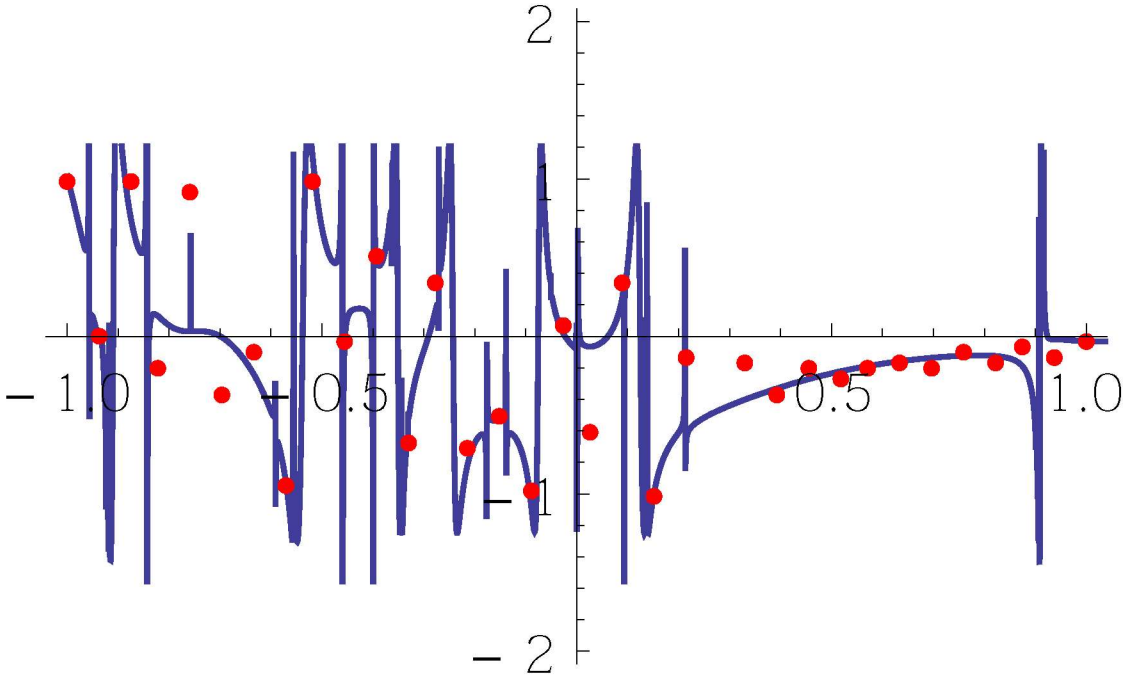



Figure 21: Second record in *ionosphere.arff* (red dots) and generated AP function

# 10 Conclusions

The aim of master thesis is to implement the AP on CUDA, what I have successfully done. I obtain relevant results from tests. Both PRNG and deterministic chaos are suitable for running with parallel implementation of AP. The use of deterministic chaos seems to be a bit better. This is given by the fewer and simpler math operations needed to get a value. CUDA is not designed to run as long kernels, as we implemented, but there is a lot of room for future optimization of these kernels. We use lots of registers in the kernel, so performance falls from the ideal. In feature CUDA architecture, kernel under kernel will be probably run, which should bring a large improvement in performance. We want to compare our results with another parallel approach in the near future. On graphics hardware it will be Opencl, and OpenPM as a CPU variant.

In classification part I describe and successfully tested tools, which implemented RDF. These tools seems to be a very good choice for deployment in classification over very big databases, because many tests shown their advantages over high dimensionality of datasets, as I describe in this work. I made also proof, that my AP algorithm generate reasonable results, when was the inserted measure derived from my generated function from *ionosphere.arff* right classified as *b* type spectra.

I see feature improvement of this work in their apply on big spectra database, where can be my AP implementation used for modelling characteristic mathematical function of spectra class.

bc. Peter Drábik

# 11 References


[1] Thizzy. 2008. Be stars. [cit. 2014-05-04] Available at: `http://www.shelyak.com/dossier.php?id_dossier=24`

[2] Rivinius T, Carciofi A, Martayan C. 2013. Classical Be stars. The Astronomy and Astrophysics Review, 1–86, doi=10.1007/s00159-013-0069-0.

[3] Virtual Observatory Architecture Overview Version 1.0[online]. last revision 14.06.2004 [cit. 2014-04-25] Available at: `http://www.ivoa.net/documents/Notes/IVOArch/IVOArch-20040615.html`

[4] Wells D C, Greisen E W, Harten R H. 1981. "FITS: A Flexible Image Transport System". *Astronomy and Astrophysics Supplement Series* 44:363–370.

[5] Schlesinger B. 1997. A Users Guide for the Flexible Image Transport System.

[6] Bramer M. 2007. Principles of Data Mining (Undergraduate Topics in Computer Science).Springer-Verlag New York, Inc.

[7] Han J, Kamber M. 2006. Data mining: concepts and techniques. The Morgan Kaufmann series in data management systems. Elsevier.

[8] Caruana R, Karampatziakis N, Yessenalina A. 2008. An empirical evaluation of supervised learning in high dimensions. ICML, pp. 96–103.

[9] Caruana R, Niculescu-Mizil A. 2006. An empirical comparison of supervised learning algorithms. ICML, pp. 161–168.

[10] Albert J. 2007. Implementation of the random forest method for the imaging atmospheric cherenkov telescope magic. Technical report, September.

[11] Breiman L, Last M, Rice J. 2003. Random forests: finding quasars. Statistical Challenges in Astronomy, pp. 243–254.

[12] T K Ho. 1995. Random decision forests. In *Document Analysis and Recognition, Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, pp. 278–282 vol.1.

[13] Breiman L. 1999. Random Forests–Random Features. Technical Report, september.

[14] Zelinka I. 2002. Artificial Intelligence in The Problems of Global Optimization (Czech Ed.) BEN, 190 p. ISBN 80-7300-069-5.

[15] Gämperle R, Müller S, Koumoutsakos P. 2002. A parameter study for differential evolution, Advances in intelligent systems, fuzzy systems, evolutionary computation, WSEAS Press, Interlaken, Switzerland, pp. 293–298

[16] Rönkkönen J, Kukkonen S. 2005. Price KV. Real-parameter optimization with differential evolution. In: Proceedings of IEEE congress on evolutionary computation, IEEE Computer Society, Washington, DC, vol. 1; pp. 506–13.

[17] Ali W, Mohameda H Z S, Tareq A E. 2013. Real pa-rameter optimization by an effective differential evolution algorithm. Faculty of Computers and Information, Cairo University DOI=10.1016/j.eij.2013.01.001 http://dx.doi.org/10.1016/j.eij.2013.01.001

[18] Godfrey C, Onwubolu D D. 2009. Differential Evolution: A Hand-book for Global Permutation-Based Combinatorial Optimization, Springer, ISBN: 3540921508, pp.192

[19] Shemyakin V. 2012. Differential Evolution approach and parameter estimation of chaotic dynamics. Lappeenranta, Master's thesis at Lappeenranta University of Technology Faculty of Technology Department of Mathematics and Physics.The thesis was supervised by: Prof. Heikki Haario.

[20] Zelinka I, Lampine J. 2000. SOMA—Self-Organizing Migrating Algorithm, in *Proceedings of the 6th International Conference on Soft Computing*, Brno, Czech Republic, pp. 177–187.

[21] Zelinka I, Lampinen J, Noulle L. 2001. On the theoretical proof of convergencefor aclass of SOMA search algorithms, in Proceedings of 7th International Conference on Soft Computing, Brno,Czech Republic, pp. 103–110.

[22] Zelinka I, Lampinen J, Nolle L. 2001. On the theoretical proof of convergencefor a class of SOMA search algorithms, in Proceedings of the 7th International Mendel Conference on Soft Computing, Brno, Czech Republic, pp. 103–110.

[23] Zelinka I. 2004. "SOMA—Self organizing migrating algorithm," in *New optimization techniques in engineering*, G. C. Onwubolu and B. V. Babu, Eds. Berlin, Germany: Springer, ch. 7.

[24] Zelinka I. 2009. "Real-time deterministic chaos control by means of selected evolutionary techniques" in *Engineering Applications of Artificial Intelligence 22(2)*, pp. 283-297.

[25] Nolle L, Zelinka I, Hopgood A A, Goddyear A. 2005. Comparison of a self-organizing migration algorithm with simulated annealing and differential evolution for automated waveform tuning, *Advances in Engineering Software*, vol. 36, no. 10, pp. 645–653.

[26] Kadlec P, Raida Z. 2011. "A novel multi-objective self-organizing migrating algorithm," Radioengineering, vol. 20, no. 4, pp. 77–90.

[27] Varacha P, Pospisilik M, Motyl I, Bliznak M, Slovak D, Krampl J, Kolek J. 2013. Competitive evaluation of selected evo-lutionary algorithms and SOMA. *International Journal of Mathematics and Computers in Simulation*. vol. 7, iss. 1, pp. 42-49. ISSN 1998-0159.

[28] Davendra D, Zelinka I, Onwubolu G. 2010. Chaotic attributes and per-mutative optimization. Zelinka, Ivan (ed.) et al., Evolutionary algorithms and chaotic systems. Berlin: Springer (ISBN 978-3-642-10706-1/hbk; 978-3-642-10707-8/ebook). Studies in Computational Intelligence 267, 481-517.

[29] Koza R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA.

[30] Ryan C, Collins J J, M O'Neill. 1998. Grammatical Evolution: Evolving Pro-grams for an Arbitrary Language, in Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming.

[31] O'Neill M, Ryan C. 2003. Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language : Kluwer Academic Publisher.

[32] Zelinka I, Oplatkova Z, Nolle L. 2005. "Analytic Programming – Symbolic Regression by Means of Arbitrary Evolutionary

[33] Lampinen J, Zelinka I. 1999. New Ideas in Optimization - Mechanical Engineering Design Optimization by Differential Evolution. Volume 1. London : McGraw-Hill. pp. 20. ISBN 007-709506-5.

[34] Zelinka I. Symbolic regression - an overview. [cit. 2014-05-04] Available at: `http://www.mafy.lut.fi/EcmiNL/older/ecmi35/node70.html`

[35] Zelinka I, Davendra D, Senkerik R, Jasek R, Oplatkova Z. 2011. Analytical Programming - a Novel Approach for Evolutionary Synthesis of Symbolic Structures, Evolutionary Algorithms, Prof. Eisuke Kita (Ed.), ISBN: 978-953-307-171-8, InTech, DOI: 10.5772/16166. Algorithms" In: *Special Issue on Inteligent Systems, International Journal of Simulation, Systems, Science and Technology*Volume 6, Issue 9, pp 44 – 56, ISSN 1473-8031.

[36] Cuda Toolkit Documentation v6.0. [cit. 2014-05-04] Available at: `http://docs.nvidia.com/cuda/index.html#axzz30n4Z1aWM`

[37] Kirk D, Hwu W-M. 2010. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann.

[38] [14] Rob F. 2011. CUDA Application Design and Development (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[39] Sanders J, Kandrot E. 2010. "CUDA by Example: An Introduction to General-Purpose GPU Programming", Addison Wesley. ISBN-13: 978-0-13-138768-3.

[40] Qin A K, Raimondo F, Florence F, Yew-Soon Ong. 2012. An improved CUDA-based implementation of differential evolution on GPU. GECCO: 991-998.

[41] Pavlech M, Seckar Jan. 2012. Increasing the performance of AllToAll variant of self-organizing migration algorithm using CUDA. [cit. 2014-04-25] Available at: http://www.wseas.us/e-library/conferences/2012/Sliema/MACMESE/MACMESE-51.pdf

[42] De Veronese L P, Renato K. 2010. "Differential evolution algorithm on the GPU with C-CUDA." Evolutionary Computation (CEC), IEEE Congress on. IEEE, 2010.

[43] Kralj P. 2013. Differential Evolution with parallelised objective functions using CUDA. [cit. 2014-05-04] Available at: http://www.codehunter.eu/media/Kralj_Differential_Evolution_with_parallelised_objective_functions_using_CUDA.pdf

[44] cuRAND. CUDA Toolkit Documentation [cit. 2014-05-04] Available at: http://docs.nvidia.com/cuda/curand/#axzz30n4Z1aWM

[45] Hilborn R C. 2000. Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers, Oxford University Press, 2000, ISBN: 0-19-850723-2.

[46] May R M. 2001. Stability and Complexity in Model Ecosystems, Princeton University Press, ISBN: 0-691- 08861-6.

[47] Kominkova Z, Senkerik R, Zelinka I, Pluhacek M. 2013. Analytic programming in the task of evolutionary synthesis of a controller for high order oscillations stabilization of discrete chaotic systems. *Computers & Mathematics with Applications* 66(2), pp. 177-189.